Rationale Management
Chapter 12

An aircraft example

A320

- First fly-by-wire passenger aircraft
- 150 seats, short to medium haul

A319 & A321

- Derivatives of A320
- Same handling as A320

Design rationale

- Reduce pilot training & maintenance costs
- Increase flexibility for airline

An aircraft example (2)

A330 & A340

- Long haul and ultra long haul
- 2x seats, 3x range
- Similar handling than A320 family

Design rationale

- With minimum cross training, A320 pilots can be certified to fly A330 and A340 airplanes

Consequence

- Any change in these five airplanes must maintain this similarity

Overview: rationale

- What is rationale?
- Why is it critical in software engineering?
- Centralized traffic control example
- Rationale in project management
  - **Consensus building**
  - **Consistency with goals**
  - **Rapid knowledge construction**
- Summary

What is rationale?

Rationale is the reasoning that lead to the system.

Rationale includes:

- the ***issues*** that were addressed,
- the ***alternatives*** that were considered,
- the ***decisions*** that were made to resolve the issues,
- the ***criteria*** that were used to guide decisions, and
- the ***debate*** developers went through to reach a decision.

- Rationale can be used at any phase of the development lifecycle but the focus is generally on system design

Levels of Rationale Capture

- No explicit rationale capture
  - **Resources spent only on development, rationale present only in developer's memories and in records such as email, memos, etc.**
- Rationale reconstruction
  - **Resources are spent recovering design rationale during documentation. Discarded alternatives and argumentation are generally not captured.**
- Rationale capture
  - **Major effort is spent in capturing rationale as decisions are made. Rationale information is documented as a separate model and cross-referenced with other models.**
- Rationale integration
  - **The rationale model becomes the central model developers use as a live and searchable information base. The system models represent the sum of the decisions captured in the information base.**

Why is rationale important in software engineering?

Many software systems are like aircraft:

They result from a large number of decisions taken over an extended period of time.

- Evolving assumptions
- Legacy decisions
- Conflicting criteria

-> high maintenance cost

-> loss & rediscovery of information

Uses of rationale in software engineering

- Improve design support
  - **Avoid duplicate evaluation of poor alternatives**
  - **Make consistent and explicit trade-offs**
- Improve documentation support
  - **Makes it easier for non developers (e.g., managers, lawyers, technical writers) to review the design**
- Improve maintenance support
  - **Provide maintainers with design context**
- Improve learning
  - **New staff can learn the design by replaying the decisions that produced it**

Representing rationale: issue models

Issue modeling of dialectic activity is the most promising approach so far:

- More information than documents: captures trade-offs and discarded alternatives that design documents do not.
- Less messy than communication records: communication records contain everything.

Issue models represent arguments in a semi-structure form:

- Nodes represent argument steps
- Links represent their relationships

ATM Example

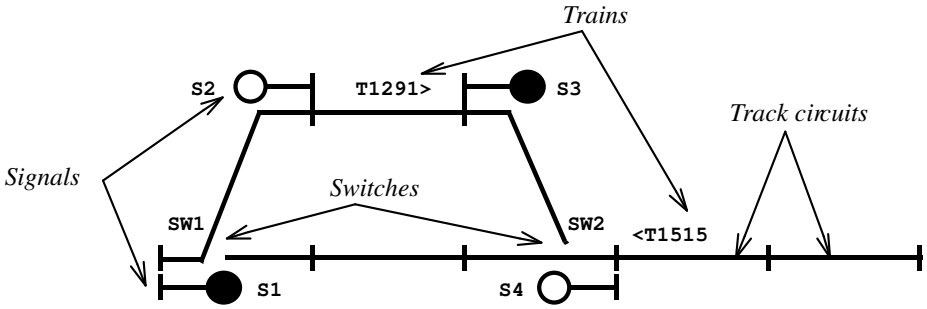
Question: Alternative Authentication Mechanisms?

References: Service: Authenticate

Decision: Smart Card + PIN

	Criteria 1: ATM Unit Cost	**Criteria 2:** Privacy
Option 1: Account number	+	–
Option 2: Finger print reader	–	+
Option 3: Smart Card + PIN	+	+

Centralized traffic control



- CTC systems enable dispatchers to monitor and control trains remotely
- CTC allows the planning of routes and re-planning in case of problems

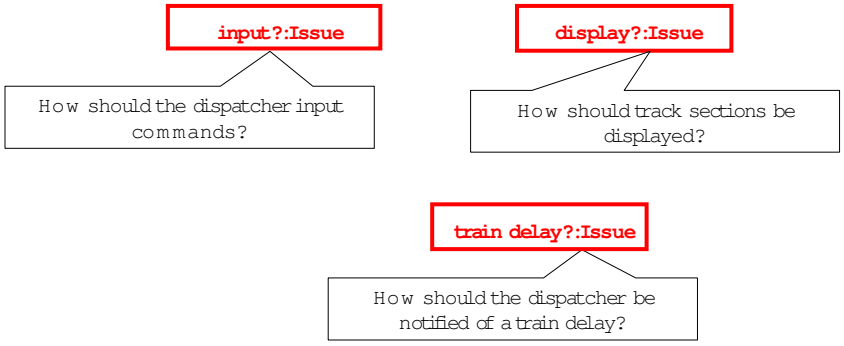
Centralized traffic control (2)

CTC systems are ideal examples of rationale capture:

- Long lived systems (some systems include relays installed last century)
  - **Extended maintenance life cycle**
- Although not life critical, downtime is expensive
  - **Low tolerance for bugs**
  - **Transition to mature technology**

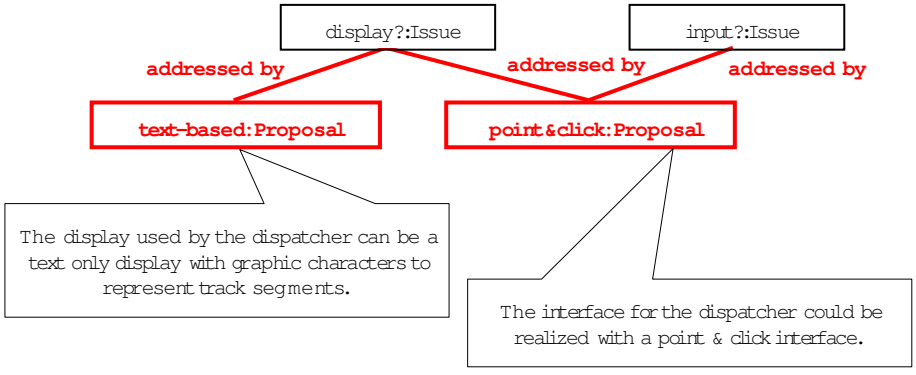
Issues

- Issues are concrete problem which usually do not have a unique, correct solution (so called "wicked" problems).
- Issues are phrased as questions. Should focus only on the problem, not on possible alternatives to address it.



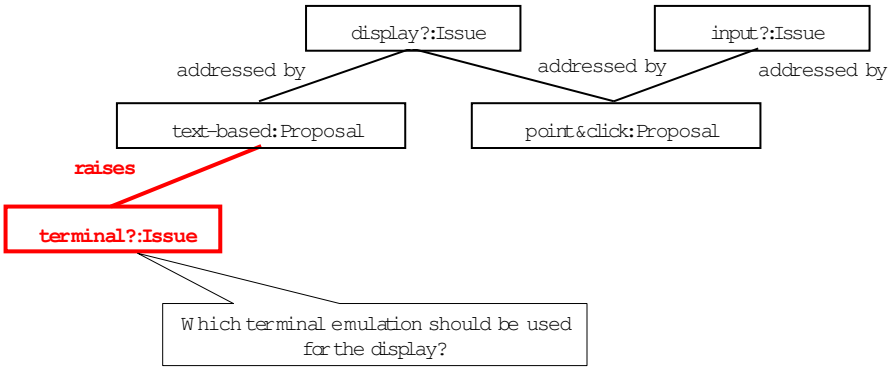
Proposals

- Proposals are possible solutions to issues.
- One proposal can be shared across multiple issues.
- Proposals enable developers to explore the solution space thoroughly.



Consequent issue

- Consequent issues are issues raised by the introduction of a proposal.

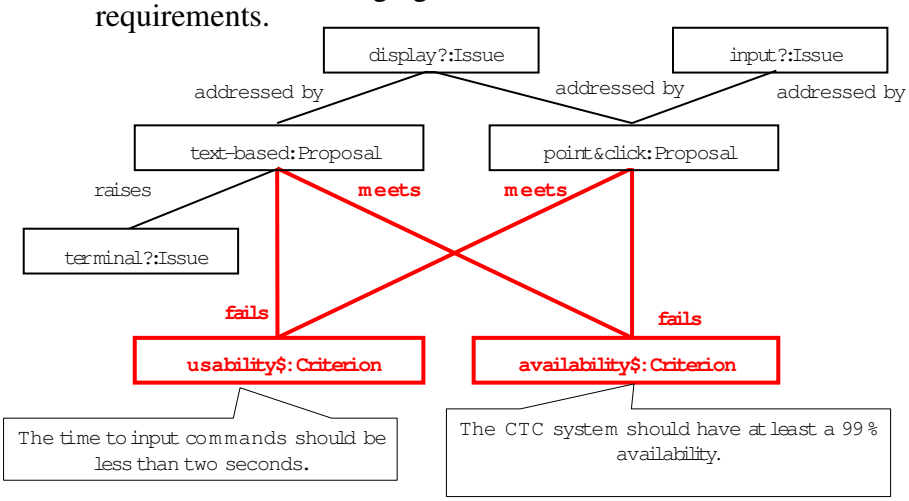


Proposals

- A proposal should only contain information related to the solution, not its value, advantages, and disadvantages
  - **These are addressed by criteria and arguments**
- A proposal need not be a good or valid answer
  - **This allows developers to explore the solution space thoroughly**
  - **Proposals are used to represent the solution to the problem as well as discarded alternatives**

Criteria

- A criteria represent a goodness measure.
  - **Not to be confused with an argument or issue**
- Criteria are often design goals or nonfunctional requirements.



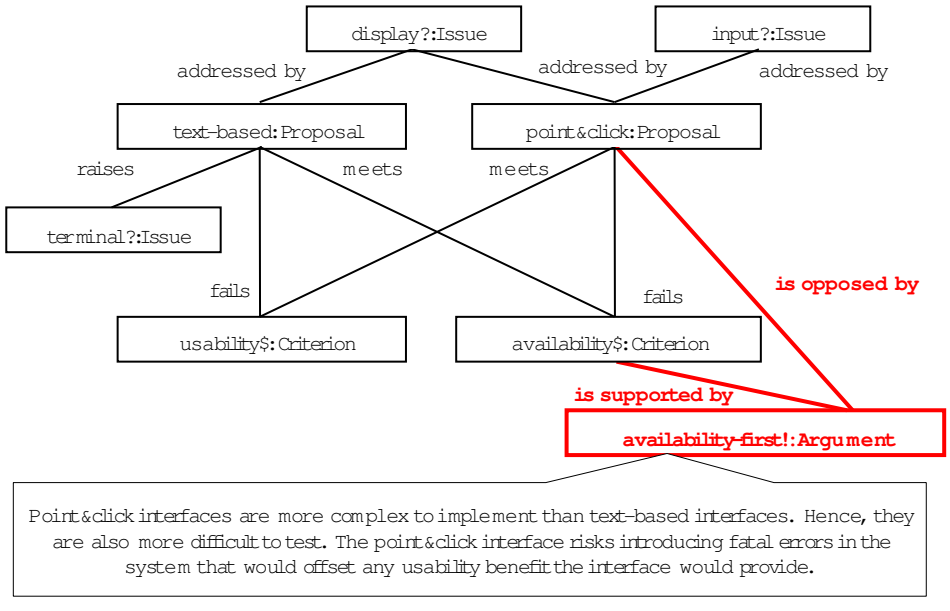
Criteria

- Associations linking proposals and their criteria may represent a trade-off
  - **In our example, each proposal maximizes one of the two criteria**
  - **The issue is to decide which criteria has a higher priority, or find a new proposal**

Arguments

- Arguments represent the debate developers went through to arrive to resolve the issue.
- Arguments can support or oppose any other part of the rationale.
- Arguments constitute the most part of rationale.

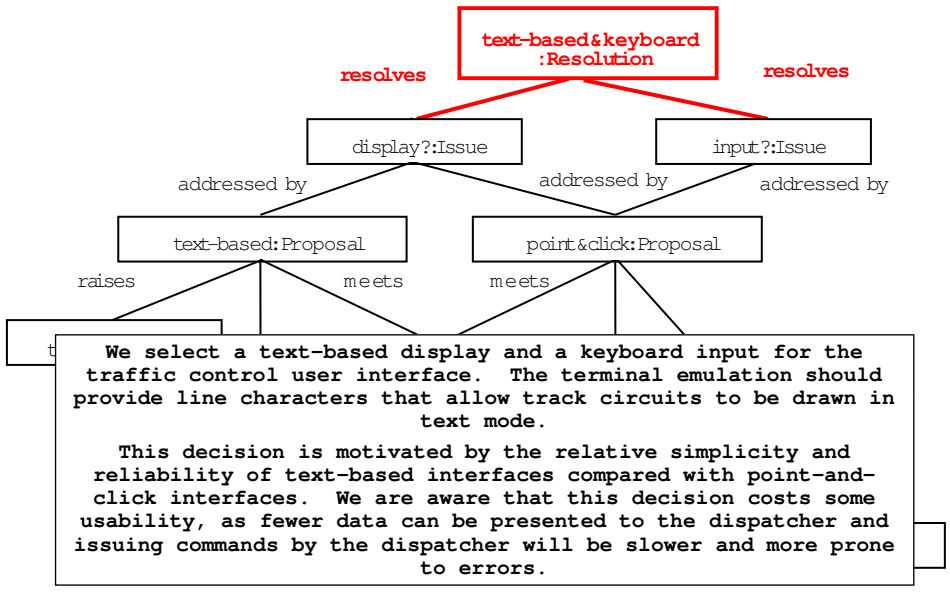
Arguments (2)



Resolutions

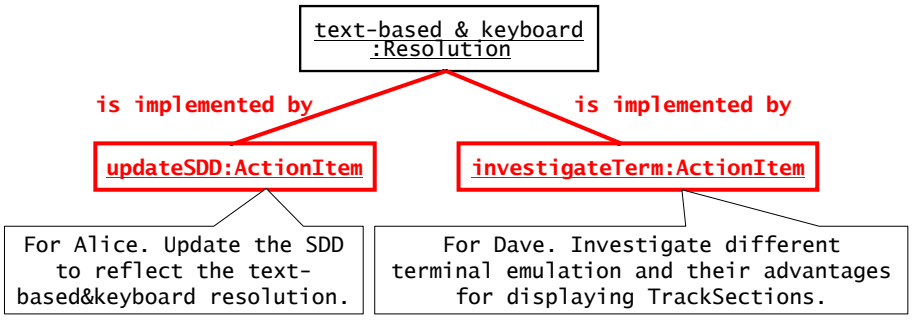
- Resolutions represent decisions.
- A resolution summarizes the chosen alternative and the argument supporting it.
- A resolved issue is said to be closed.
- A resolved issue can be re-opened if necessary, in which case the resolution is demoted.

Resolutions (2)



Implementing Resolutions

- A resolution is implemented in terms of one or more action items
  - **A person is assigned to an action item with a completion date**



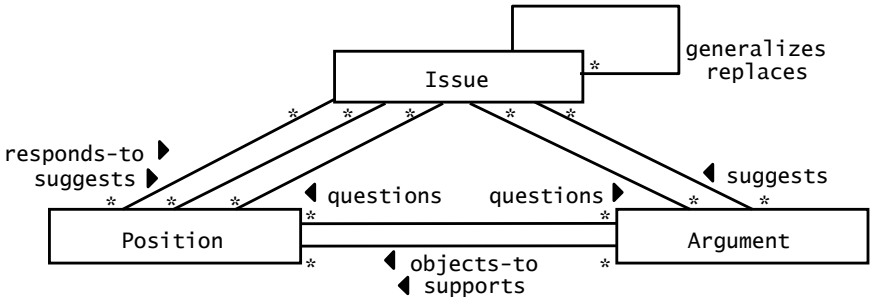
Issue-Based Models and Systems

- We've just discussed a general approach to capturing rationale using a UML style syntax

- Several models have been proposed to capture rationale
  - **IBIS – Issue Based Information System (Kunz & Rittel, 1970)**
  - **DRL - Decision Representation Language (Lee, 1990)**
  - **QOC – Questions, Options, and Criteria (MacLean, 1991)**
  - **NFR Framework (Chung, 1999)**

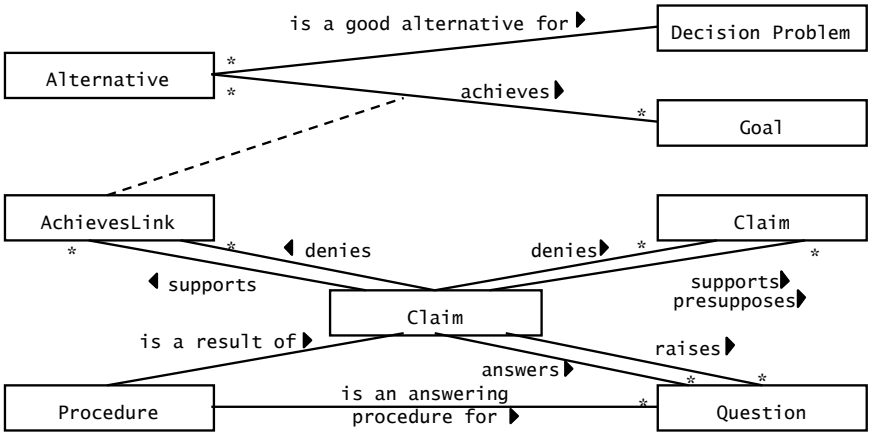
IBIS Model

- Three nodes, but seven types of links
- Did not originally include Criterion or Resolution



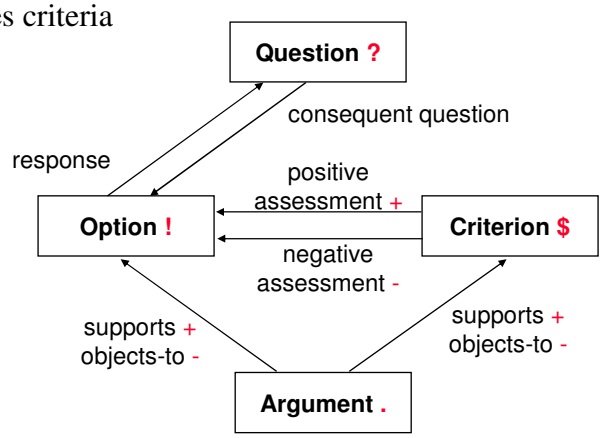
Decision Representation Language

- Seven types of nodes and fifteen types of links



Questions, Options, Criteria

- Designed for capturing rationale after the fact (e.g., quality assessment).
- QOC emphasizes criteria



Capturing Rationale in Meetings

- Needs a dedicated person to take minutes
- Publish agenda prior to meeting

Agenda Example

AGENDA: Integration of access control and notification

1. Purpose
The first revisions of the hardware/software mapping and the persistent storage design have been completed. The access control model needs to be defined and its integration with the current subsystems, such as `NotificationService` and `TrackingSubsystem`, needs to be defined.

2. Desired outcome
Resolve issues about the integration of access control with notification.

3. Information sharing [Allocated time: 15 minutes]
`AI[1]:` Dave: Investigate the access control model provided by the middleware.

4. Discussion [Allocated time: 35 minutes]
`I[1]:` Can a dispatcher see other dispatchers' `TrackSections`?
`I[2]:` Can a dispatcher modify another dispatchers' `TrackSections`?
`I[3]:` How should access control be integrated with `TrackSections` and `NotificationService`?

5. Wrap up [Allocated time: 5 minutes]
Review and assign new action items.
Meeting critique.

Chronological Minutes Example

CHRONOLOGICAL MINUTES: Integration of access control and notification

4. Discussion
...
`I[3]:` How should access control be integrated with `TrackSections` and `NotificationService`?

Dave:	The `TrackSection` maintains an access list. The notification service asks the `TrackSection` about who has access.
Alice:	We should probably reverse the dependency between `TrackSection` and `NotificationService`. Instead, the `UIClient` requests subscriptions from the `TrackSection`, which checks for access and then calls the `NotificationService`. This way, all protected methods are in one place.
Dave:	This way the `TrackSection` can also more easily unsubscribe dispatchers when their access is revoked.
Ed:	Hey, no need for access control in `NotificationService`: Dispatchers can see all `TrackSections`. As long as the `NotificationService` is not used for changing the `TrackSection` state, there is no need to restrict subscriptions.
Alice:	But thinking about the access control on notification would be more general.
Ed:	But more complex. Let's just separate access control and notification at this point and revisit the issue if the requirements change.
Alice:	Ok. I'll take care of revising the `TrackingSubsystem` API.

...

Structured Minutes Example

- Produced after review of chronological minutes and distributed to all parties

4. Discussion

...

`I[3]:` How should access control be integrated with `TrackSections` and `NotificationService`?

`P[3.1]:` `TrackSections` maintain an access list of who can examine or modify the state of the `TrackSection`. To subscribe to events, a subsystem sends a request to the `NotificationService`, which in turns sends a request to the corresponding `TrackSection` to check access.

`P[3.2]:` `TrackSections` host all protected operations. The `UIClient` requests subscription to `TrackSection` events by sending a request to the `TrackSection`, which checks access and sends a request to the `NotificationService`.

`A[3.1]` for `P[3.2]:` Access control and protected operations are centralized into a single class.

`P[3.3]:` There is no need to restrict the access to the event subscription. The `UIClient` requests subscriptions directly from the `NotificationService`. The `NotificationService` need not check access.

`A[3.2]` for `P[3.3]` Dispatchers can see the state of any `TrackSections` (see `R[1]`).

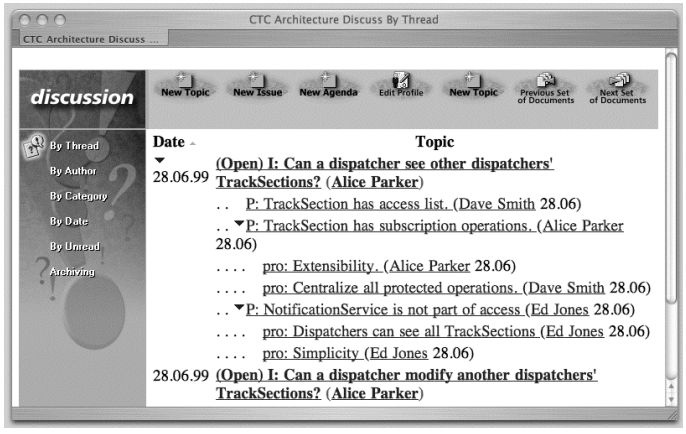
`A[3.3]` for `P[3.3]`: Simplicity.

`R[3]: P[3.3].` See action item `AI[2]`.

...

Asynchronous Rationale Capture

- Can also capture rationale asynchronously via issue database
- Developers can access and post issues, proposals, arguments, and resolutions with web forms
- Can be combined with real-time capture; structured minutes results in creation of issues in the issue database



Overview: rationale

- What is rationale?
- Why is it critical in software engineering?
- Centralized traffic control example
- Rationale in project management
  - Consensus building (WinWin)
  - Consistency with goals
  - Rapid knowledge construction
- Summary

Consensus building

Problem

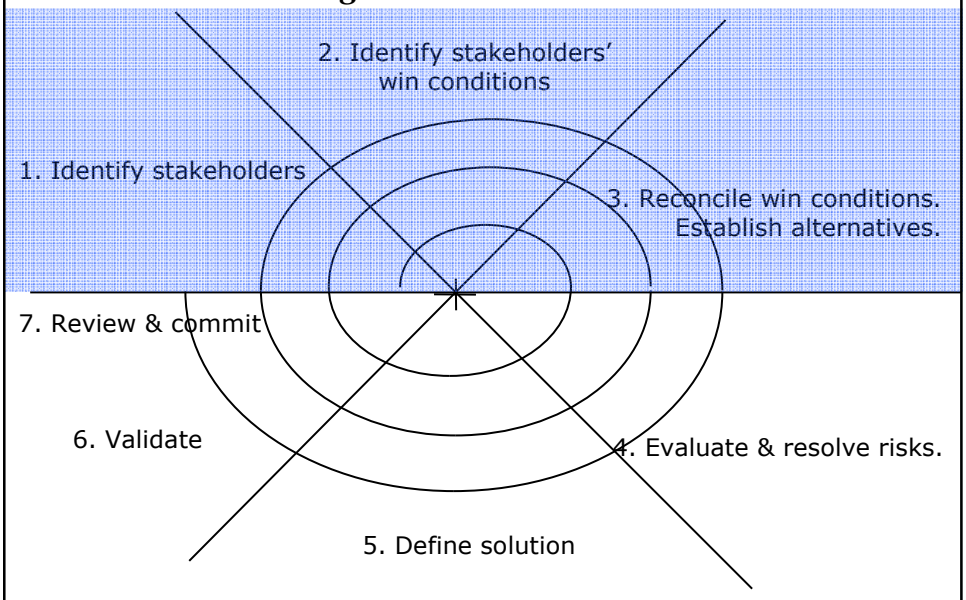
- Any realistic project suffers the tension of conflicting goals
  - **Stakeholders come from different background**
  - **Stakeholders have different criteria**

  - **Client: business process (cost and schedule)**
  - **User: functionality**
  - **Developer: architecture**
  - **Manager: development process (cost and schedule)**

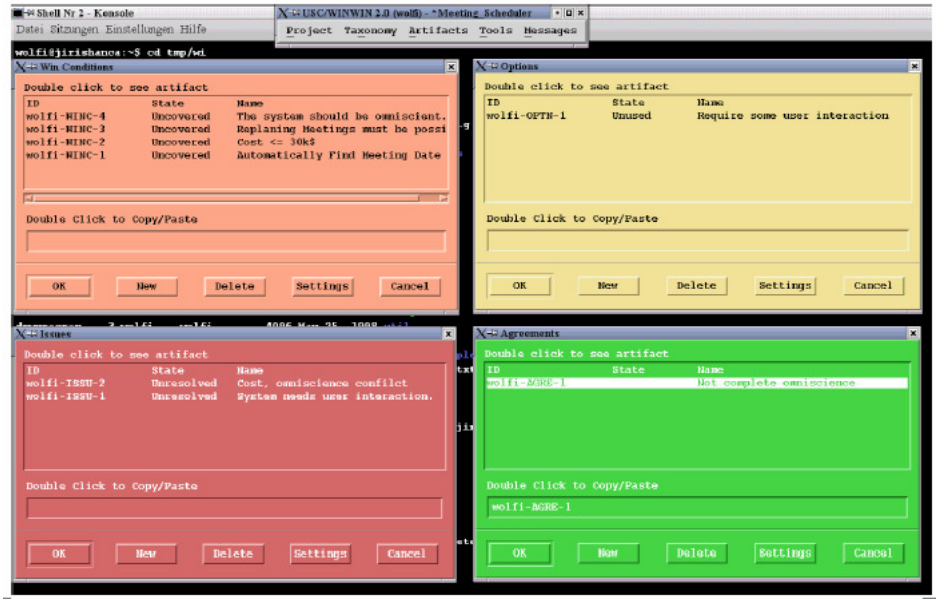
Consensus building: WinWin

- Incremental, risk-driven spiral process
  - **Identification of stakeholders**
  - **Identification of win conditions**
  - **Conflict resolution**
- In meetings or asynchronous groupware tool
  - **Stakeholders post win conditions**
  - **Facilitator detects conflict**
  - **Stakeholders discuss alternatives**
  - **Stakeholders make agreements**

Consensus building: Process



Consensus building: WinWin tool



Harvard Method of Negotiation

- Traditional negotiation
  - **Defend one's position, cite advantages, denigrate other's position citing its disadvantages**
  - **Time consuming and moves in small steps toward consensus**
- Harvard Method
  - **Attempt to move more quickly to consensus by avoiding credibility issues and allow people to change positions**

Harvard Method of Negotiation

- Separate developers from proposals
  - **Developers can spend a lot of time developing a proposal to the point that criticism is taken as personal criticism**
  - **Can separate by putting multiple developers on the same proposal or all concerned parties participate**
  - **Separate design and implementation work**
  - **Ensure negotiation occurs before implementation**
- Focus on criteria, not on proposals
  - **Developers develop proposals with criteria in mind; their criteria may be addressed, but not necessarily other developers' criteria.**
  - **Making criteria explicit exposes the root of conflicts and allow a compromise to be negotiated**
- Take into account all criteria instead of maximizing a single one
  - **Criteria reflect different interests or parties; picking one over others alienates the other participants**

Conflict Resolution Strategies

- Majority Wins
  - **Majority vote decides deadlock**
  - **Assumes the opinion of each participant matters equally and that statistically the group usually makes the right decision**
- Owner has last word
  - **The person who raised the issue or has the largest stake is responsible for deciding the outcome**
- Management is always right
  - **Fall back on the org hierarchy and let the manager impose a decision**
- Expert is always right
  - **External or third party expert makes the decision**
- Time decides
  - **Leave issue unresolved and time pressure forces a decision**

Conflict Resolution Strategies

- Majority Wins
  - **Generally does not work well; inconsistent results and decisions not well supported by the rest of the participants**
- Owner has last word
  - **Generally does not work well; inconsistent results and decisions not well supported by the rest of the participants**
- Management is always right
  - **Generally leads to better technical decisions and better consensus when the manager is sufficiently knowledgeable**
- Expert is always right
  - **Generally leads to better technical decisions and better consensus when the manager is sufficiently knowledgeable**
- Time decides
  - **Fallback; may result in costly rework or disaster**

In practice, first attempt to reach consensus, fall back on an expert or management strategy; if fails, let time decide or take a majority vote.

Summary

- Rationale important to capture for long-term project maintenance and future development
- Consensus techniques to negotiate resolutions for rationale issues
- Major Challenges
  - **Technical: Rationale models are large and difficult to search and integrate into the development process**
  - **Non-technical: Rationale is an overhead that benefits mainly other participants**
- Some successful cases have illustrated the importance of tightly integrating the capture and use of rationale within a specific process (Boehm, Dutoit & Paech with REQuest)