# Patterns for designing in teams

## How teams can improve the design process

Charles Weir

## Introduction

This paper discusses some techniques to use teams effectively in software design.

Few sources appear to cover this subject. Most books on software design or methods appear to assume a single designer, possibly kept from wilder excesses by a system of reviews. However software development is almost invariably a team process. How then can we leverage the individual strengths of each development team member to get a better result than one single designer could achieve alone?

The purpose of design is to agree what will be built. Designs must be written down to get agreement[1], so in practice the design will exist as a set of written documents. According to the needs of the design process these may be anything from a collection of sketches to an encyclopaedic set of formal specifications. It is difficult yet vital to produce such documents as a team, since the design process takes much of the creative manpower in the project.

We have identified five patterns that describe some of the most effective techniques for this purpose. These are:

| | |
|---|---|
| **Multiple Competing Designs** | This uses designers working independently to generate a variety of design ideas. |
| **Decision Document** | This uses a document and a facilitator to compare and contrast two different designs. |
| **Creator-Reviewer** | This uses different skills available to provide effective feedback to designers |
| **Ad-hoc Corrections** | This uses a 'corrections' copy of a design document to co-ordinate design changes by different members of the implementation team. |
| **Master-Journeyman** | This divides a large development into manageable components to allow many design teams to work in parallel. |

To provide an idea how the patterns interrelate, figure 1 shows how the patterns might apply within the lifetime of a design.

[1] To quote Goldberg (see [Goldberg-Rubin]): "If it's not written down, it doesn't exist".

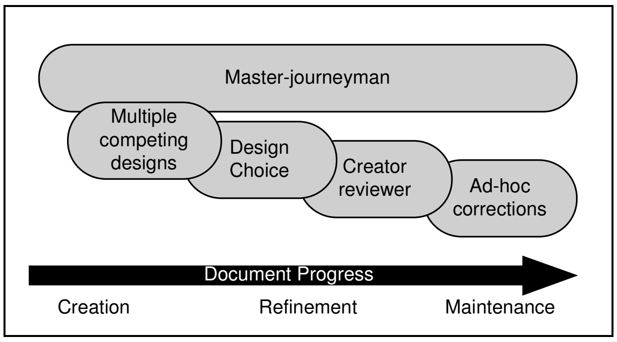

*Figure 1: Pattern Interactions*

These patterns derive from the author's experience as an OO design consultant working with a variety of team projects in banking, telecommunications and industry. Many of the examples of use come from this experience. The set of patterns is by no means complete; there are other related patterns ([Cockburn] for example) and relationships to patterns in project management such as [Coplien] and [Harrison].

Each description contains a *role diagram* showing the roles involved and their interactions. An individual may change roles, or undertake more than one role.

The patterns use the word *designer* to mean anyone sufficiently skilled to carry out the job of design alone. Some sources ([Coplien], for example) use the term 'architect' with the same meaning. Of course in practice designers may receive assistance from mentors or others in the team. The role descriptions ignore this additional complexity.

Also outside the scope of these patterns is the effect of dysfunctional teams. We assume the team members to be co-operative, intelligent and constructive in their working together. Dysfunctional teams will be less effective in using these patterns.

---

## Pattern 1: Multiple Competing Designs

In this pattern several designers generate a variety of design ideas, the best of which are then incorporated in a single design.

### The Problem

A team can think itself into a cul-de-sac.

A single person or coherent team can easily get stuck on a single design solution, and fail to see alternatives that may be more appropriate to the particular situation. Once the team has reached one conclusion, the combined effects of ego and inertia make it very difficult to consider any other possibilities.

### Solution

Working individually or in pairs, several designers produce a different design for the same item. The design should be sufficiently detailed to establish the major issues and major choices for the design, but not have every aspect completed. A reasonable maximum time spent working on the design might be a day; a minimum might be half an hour. It's a good idea, though, to allow a longer elapsed time to allow ideas to mature.

The entire team of all the designers then evaluates each design in a series of reviews. It is vital that the reviews are constructive; there is a moderator to ensure this. In particular the review should establish for each design:

- A list of significant strengths of the design

- A set of circumstances which would favour that design over any of the others
- A list of the specification questions raised by this design that the team may need to resolve for the project as a whole.

Finally, a lead designer combines the strengths of each design in a single design using the information extracted from the reviews.

## Roles Involved

This pattern requires at least two people capable of completing the design as it stands. Since it requires all the designers to participate in a meeting, a reasonable maximum would be six designers working in pairs. One of the design team or a further team member receives the role of lead designer. In addition it needs an independent moderator.

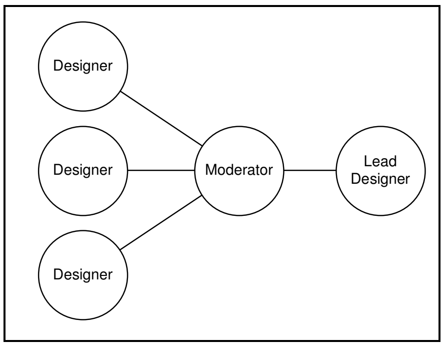

*Figure 2:Roles in the Multiple Competing Design pattern*

## Forces

It is vital to the success of this approach that the lead designer ‘plays fair’ and avoids undue bias or ego in the choice of design ideas. Also the demands on the designer roles mean that this approach best suited only to experienced teams with a number of experienced designers.

It is possible to try having a single team produce multiple designs as a group. In practice this does not work as well as we would hope. There are two psychological traits we all share that mitigate against it. The first is the desire for *closure*, to see a single completed thing rather than outstanding work in hand. The second is our avoidance of *cognitive dissonance*, of holding conflicting ideas at the same time. [Weinberg] discusses these issues in more detail.

There are costs associated with this pattern, too. There is the time taken to generate the alternative designs, to briefly document and to discuss them. In addition there is a danger that if the synthesis process is not handled tactfully, individual designers may resent the rejection of ‘their’ designs. And this dissension may cause problems later in the project.

In practice, however, the situation of having multiple designs is a common one in projects. Individual developers and factions will create their own ideas of the best designs whether or not there is a convention encouraging it. It is important that this diversity be seen as a positive feature as described, so that the differences generate a better finished solution.

## Example

This example shows extracts from two designs for the OMT object diagram of part of a sales administration system.

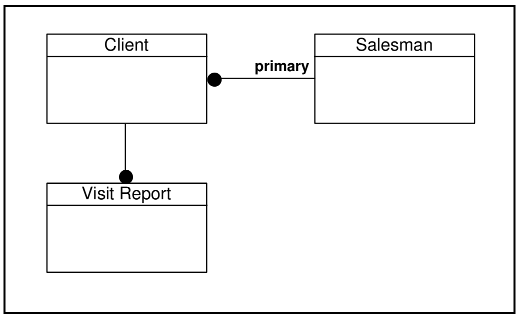

*Figure 3: Sales Administration System Design*

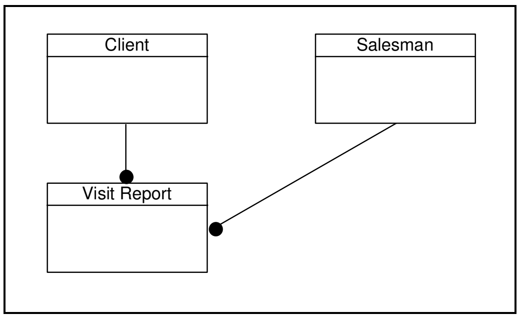

*Figure 4: Alternative System Design*

A constructive evaluation of the first design might give it as the best solution when:

- The system needs to know which salesperson is responsible for each client, rather than who made a specific visit.

Similarly the second might be appropriate when:

- The system needs to know which sales representative made each specific visit.

Thus, unless other information is available to make the decision, the comparison reveals a question to ask about the specification: which of the two is appropriate - or is a combination necessary of both?

### Related Patterns

Pattern 2: Decision Document provides a good format and procedure for choosing the best elements from the competing designs.

Pattern 3: Creator-Reviewer is appropriate to take the results of the lead designer's work. Since all of the team members are now familiar with the problem, they are in a good position to review the design actually produced.

---

## Pattern 2: Decision Document

This pattern uses a facilitator and text document to chose between or extract the strengths of different designs or design approaches.

### The Problem

It is difficult to make an objective choice between design options.

It would be nice to think that software designers are cool rational professionals, who make each separate design choice using a rational process to make the best choice from all of the available options. Sadly this is seldom the reality. Instead teams typically make design choices somewhat arbitrarily.

Sometimes the design choices take place as a battle between competing designs; the option chosen is that proposed by the most powerful member or the best debater in the team. Other times, the team may accept one particularly good design, and take in all of its design choices as a single decision. Frequently this looses sight of the many other secondary choices present in other design options.

### Solution

The team meet with a facilitator and produce a single working text, a *decision document*, comparing and contrasting the two approaches. This is a short document, perhaps a couple of sides of paper. Its purpose is to provide a firm basis for a decision between the options. It therefore highlights the strengths and weaknesses of each option and analyses what set of requirements and priorities would favour one over the other.

This decision may be a clear synthesis between the two options ("combine this aspect of option A with that of option B"). Or it may be a set of tests and evaluation procedures to make the choice ("Do A if the performance text is adequate for our needs"). Or else it may be a balanced decision for the project architect or sponsors ("A will give better reliability in future for a higher cost. Do we want to pay for it?").

## Roles Involved

There are two major roles: designers represent each competing design option; the facilitator acts as an impartial moderator of the discussion.

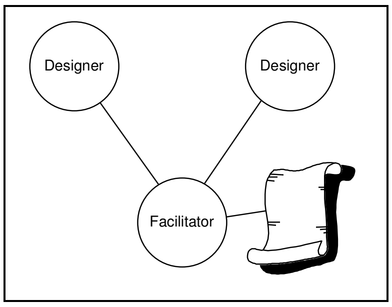

*Figure 5:Roles in the Decision Document Pattern*

The responsibilities of the designers are:

- To explain their options effectively and
- To participate in coming up with options and conclusions.

The responsibilities of the facilitator are:

- To create and write the decision document. This will initially be where all those involved with the workshop can see it.
- To encourage the participants to identify and analyse each area of dissent and divergence between the options in terms. They will aim to create a set of underlying assumptions or requirements for each option.
- To defuse antagonism and destructive discussion between the participants.

The format of the decision document may vary. We recommend a format with headings as shown below.

| | |
|---|---|
| **Options** | A brief description of each option, with a sketch diagram. |
| **Forces** | A list of considerations that may affect the choice between each option. |
| **Conclusions** | Either a straightforward conclusion ("therefore we shall do X") or a statement in objective terms of how the decision will be made ("if the performance of approach Y is adequate in a prototype, we shall do Y, else X"). |

## Forces

This approach requires a mediator who doesn't champion one approach or the other, and who has the respect of the designers involved. If no such mediator is available it's possible for one of the designers to take this role, but in this case it is clearly difficult for the process to be unbiased.

This decision process clearly requires a significant time input from the participants. For simple decisions it may simply be too expensive to put into practice.

Another problem may be an inconclusive decision; it may simply turn out that the team cannot find an objective basis to use. Even in this case the process usually adds understanding to the decision process.

The participants may see the analysis process as threatening; there is the possibility that their design be found to be 'wrong'. It is vital that the facilitator emphasises that the positive aspects of getting the best aspects of every design.

It is usually best if the first version of the decision document is hand-written and available where all the workshop participants can see it. If there are several participants, it may be best to use wall posters rather than a small sheet of paper.

The resulting document becomes part of the project documentation. As the requirements and the project implementation change, some of the project constraints and constants may change. The decision documents can help future designers understand which of the early design decisions have then become invalid; the designers may even choose to reimplement if necessary.

### Example

Take as an example the two design options for the sales administration system in Pattern 1: Multiple Competing Designs. The workshop would generate a short decision document as follows:

The problem description section contains diagrams much as shown there.

The forces section might include brief comments on such aspects as the design, performance considerations, space considerations, the implementation of persistence and distribution for each option.

Assuming that none of these are significant in this particular project the conclusions section might then read: "Do we need to know which salesperson is responsible for each client? If so, select option one, else option 2"

### Known Uses

As a software design consultant, I frequently find myself in the position of adjudicator between two or more design options. I have used this approach in a variety of different projects.

Paul Fertig has used a similar but more formal approach in work for IBM.

### Related Patterns

This pattern is particularly useful to get the best from the competing approaches in Pattern 1: Multiple Competing Designs.

## Pattern 3: Creator-Reviewer

This uses the experience and critical facilities of team members who need not be experienced designers to add value to the work of a single designer or design team.

### The Problem

People make mistakes.

It is notoriously difficult to see problems and errors in your own work. There is much research work available on the reasons behind this (see [Bezier]); all suggest that any realistic solution must involve other people. Thus if we have one or two designers producing a design alone there is a strong likelihood of undetected errors.

## Solution

The designer produces a draft or a complete design. Each of one or more reviewers receives a copy, and provides constructive feedback. There are many choices for ways to do reviews, but two are particularly effective:

| | |
|---|---|
| Meeting Review | The reviewers meet to walk through the document section by section. A 'Review leader' moderates the process; a recorder keeps track of the comments agreed. |
| Distribution Review | The document circulates to each reviewer, with a form to receive comments. Sometimes this can occur electronically using a mechanism such as Lotus Notes. |

It is important that the reviewers concentrate on constructive feedback for the document. In a meeting review it is the task of the review leader to ensure this.

There are significant benefits to doing reviews in this way. In particular it improves communication between the team members, and those whose skills are less suited to design can make a significant contribution to the design process.

## Roles Involved

This pattern requires a single designer or design team, and a number of reviewers, who need not be skilled at design.

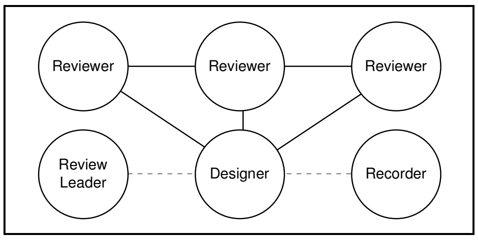

*Figure 6:Roles in the Creator-Reviewer pattern*

The review leader has the role of ensuring the effectiveness of the review. In a meeting review the leader acts as moderator; in a distribution review the leader must ensure that each reviewer contributes within a reasonable time limit. In a meeting review, it is important that there is also a recorder role who notes down each defect discovered. Fagin's inspection process (see [Gilb]) is a particularly effective format for reviews.

It is best if the reviewers are as varied as possible in skills, knowledge, and personality. Particularly good candidates for reviewers are:

- Someone with detailed knowledge of the application domain.
- A finisher (see [Belbin]) - typically someone who likes to get paperwork correct.
- Another designer who has not been involved with this design.

## Forces

There are significant costs associated with reviews. Firstly they are expensive in time and effort. Also programmers often find them less fun than normal programming or writing, so they may be demotivating to the team.

So we must be careful to keep reviews short and effective, and to limit reviews to the documents that really require them. For many documents we can achieve much of the same effect with less formality by asking a single colleague for comments.

There are many variations to the review procedures. In some, the designer may not be allowed to speak at all the review. See also [Bezier] for other possibilities.

### Related Patterns

A related pattern is Coplein's Pattern 16, "Review the Architecture" (see [Coplien]). The emphasis of that pattern is on team communication, another valuable consequence.

---

## Pattern 4: Master-Journeyman

### The Problem

We need to partition the design work for a large system.

It is essential that there is a single architect or at most a small team to provide the design integrity for the entire system[2]. Yet in a large development project it is unlikely to be possible for this core architecture team to do all of the design work.

### Solution

A *master design team* provides an overview of the system architecture and divides the system into components each of which can be a separate autonomous development. *Journeyman* architects then take on the design of each separate component, each being the chief designers of their own component of the system.

The master design team must identify and specify the items shared by all of the components. Typically these will be:

| | |
|---|---|
| **Core architecture.** | This might include the physical system description, a description of the components and their interactions, examples of interactions between them, and a description of the changes anticipated by the architecture. |
| **Architectural vision.** | This is the most difficult area to define, and one of the most important to specify. Some examples of this policy are design patterns for all the sub-projects to share, and a set of priorities for handling design trade-offs: size vs. speed, for example. |
| **Interfaces.** | This might include a description of the types of transaction across each interface and an outline of the specification. The details of the specification will typically be left to the designers responsible for each component. |
| **Specification control.** | The master team must vet any significant changes to the specification, to ensure that the overall system architecture remains consistent with the requirements. |

In addition, in a development where the components are within the same item of software, the master design team must typically specify or commission software policies common to the whole development. These might include:

- Error handling policy
- Coding guidelines
- Testing strategy
- Use of third-party libraries
- Multi-threading strategy.

2 'Designing Object Systems' discusses this point in detail (see [Cook-Daniels], last chapter).

*Journeymen propose; masters dispose*: The journeyman designers are encouraged to propose policies and changes that affect other parts of the system than their own. However the master designer retains the decision of whether and how these become part of the master plan.

## Roles Involved

There are two significant roles in this pattern: master designer and journeyman designer. Each may in practice be a team of up to three people.

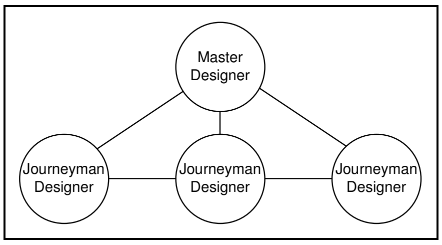

*Figure 7:Roles in the Master-Journeyman Pattern*

## Forces

A key success factor of this approach is the cleanness of the division into different autonomous developments. A design requiring too much co-ordination between any two journeyman teams will lead to conflict or misunderstandings. The division is difficult to get right first time, so we must be prepared to change it as required. One effective approach is to divide the system into domains (see [Cook-Daniels], chapter 11) and to specify the domain interfaces only.

To be workable this design requires at least one separate component for each journeyman designer. However it may be effective to have one designer take on responsibility for more than one component.

## Example

A large financial information company had a project for a new telecommunications system comprising separate developments at more than ten sites world-wide. The whole software took several hundred work years to develop. A core architecture team of two people defined the protocols and architectural vision of the system.

Then as the project got under way up and the developments started around the world, the core team members travelled between the sites ensuring the co-ordination of the developments involved. They provided a set of master design documents that they modified as required (see Pattern 5: Ad-hoc Corrections).

The initial project was a success. Over the next five years as changes occurred to the system there has continued to be a single team co-ordinating the design, and the system has kept its integrity.

## Known Uses

The name of this pattern comes from an arrangement common to many mediaeval crafts. A master craftsman would define and takes responsibility for an entire piece of work; many journeyman craftsmen would work on separate pieces, co-ordinated by the master.

Most large software projects use this approach.

## Related Patterns

A related pattern is Coplein's "Pattern 13: Architecture controls product" - see [Coplien].

## Pattern 5: Ad-hoc Corrections

### The Problem

Design documents are seldom up-to-date.

New developers coming to a project often have to depend on verbal descriptions of the changes since the original design, or to find out for themselves by reading the code. Developers already on the project find the same problem when they need to find out about unfamiliar parts they of the system. This costs time and reduces the architectural cohesion of the system.

### Solution

The team keep a master copy of the design accessible to the entire team available on paper. Anyone who needs to deviate from the design must write a correction in the margin, deletes sections that are now inappropriate, or scribble in a short description of the nature of the change.

Ultimately one team member takes responsibility for updating any machine-readable copies of the design to correspond to the corrections. This may be the designer. Alternatively the update may be a learning task for a new team member.

### Roles Involved

There may be one or more designers and developers working to a single design. Any may be in a position to identify and make corrections to the design.

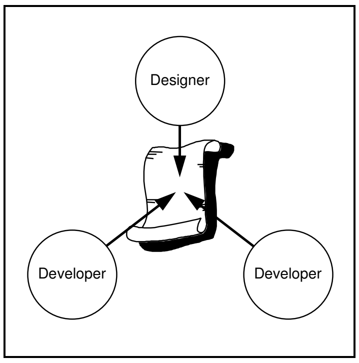

*Figure 8:Roles in the Ad-hoc Corrections Pattern*

### Forces

It may be tempting to keep the document and its annotations on-line (hence use version control and have remote access). However this approach is often less satisfactory for two reasons:

- It is key to the pattern that the corrections be added as annotations, not as changes to the document. Ownership of the document remains with the author. In many documentation systems, support for such annotations is distinctly limited. Similarly there are technical issues of ownership, locking and version control.
- With the paper approach, the amendments are being inserted only on a copy of the master document. Psychologically they are easier to add, since they can be discarded without affecting the master.

However, with good tool support both these problems are surmountable.

### Related Patterns

This pattern is particularly suitable for the master documents for a project produced by a master design team (Pattern 4: Master-Journeyman), since these are often subject to change from a variety of forces. It should be used after the initial reviews of the documents concerned (Pattern 3: Creator-Reviewer).

## References

[Weinberg] Weinberg, G. (1971) *The Psychology of Computer Programming*, Van Nostrand Reinhold ISBN 0-442-20764-6

[Belbin] Belbin R. M (1981) *Management Teams* Butterworth-Heinemann ISBN 0-7506-0253-8

[Bezier] Bezier B (1984) *Software Testing and Quality Assurance* Van Nostrand Reinhold ISBN 0-442-21306-9

[Coplien] Coplien and Schmidt (editors 1994) *Pattern Languages of Program Design*, Addison Wesley ISBN 0-201-60734-4, Chapter 13 (Coplein) *A Generative Development-Process Pattern Language*.

[Harrison] Vlissides, Coplien and Kerth, *Pattern Languages of Program Design 2,* Addison Wesley ISBN 0-201-89527-7, Chapter 21 (Harrison) *Organizational Patterns for Teams.*

[Cockburn] Vlissides, Coplien and Kerth, *Pattern Languages of Program Design 2,* Addison Wesley ISBN 0-201-89527-7, Chapter 19 (Cockburn) *Prioritizing Forces in Software Design.*

[Cook-Daniels] Cook S and Daniels J (1994) *Designing Object Systems: Object-Oriented Modelling with Syntropy*, Prentice Hall, ISBN 0-13-203860-9

[Gilb] Gilb T (1988) *Principles of Software Engineering Management*, Addison-Wesley ISBN 0-201-19246-2

[Goldberg-Rubin] Goldberg A and Rubin K (1995) *Succeeding with Objects: Decision Frameworks for Project Management*, Addison-Wesley. ISBN 0-201-62878-3