

# Trace Cards

Nik Boyd  
educery

## Abstract

A new technique for capturing requirements information on index cards is introduced, including policy cards, intelligence quality and quantity cards, and face cards. The origins, purpose and usage of each card type are discussed, and a few representative examples are provided. Some considerations regarding how trace cards may be used in conjunction with CRC cards are also offered.

## Introduction

Software developers need simple ways to capture and share knowledge. Large (4" x 6") index cards provide a convenient form factor for capturing and sharing knowledge. Index cards are convenient from several standpoints:

- They are (literally) handy.
- They are easy to use and share.
- They encourage brevity and simplicity (focus).
- They are cheap, portable, readily available, and familiar.

For these reasons, index cards can facilitate some of the most difficult aspects of software development, including not only solution design, but also problem definition. Just so, trace cards are primarily problem oriented. Trace cards offer a convenient and efficient way to capture and share knowledge about business policies, business intelligence needs, and business activities. They offer a way to keep object-oriented software designs aligned with business policies and priorities.

## Precedent and Context

During the late 1980s, Ward Cunningham and Kent Beck introduced the use of CRC (Class, Responsibility, Collaborator) cards<sup>1</sup> as a technique for capturing and sharing object-oriented design decisions. Since then, the technique has become popular among object-oriented software developers, largely due to its simplicity and effectiveness. Just as developers need a convenient way to capture and share design decisions, requirements analysts need simple ways to capture and share knowledge about business policies, business intelligence, and business activities during software development.

Language dominates software development, especially during the earliest phases of a project when the vast majority of time is spent discussing business objectives and the intended effects for a project. Ultimately, all software elements, components and composites are named, and they are often organized based on linguistic affinities, e.g., as objects and sentential expressions in object-oriented designs. So, the words used in discussions about business problems and solutions will naturally have a dramatic and durable impact on the resulting software. Beck and Cunningham emphasized this point with respect to CRC cards:

"The class name of an object creates a vocabulary for discussing a design. Indeed, many people have remarked that object design has more in common with language design than with procedural program design. We urge learners (and spend considerable time ourselves

while designing) to find just the right set of words to describe our objects, a set that is internally consistent and evocative in the context of the larger design environment."

"Responsibilities identify problems to be solved. The solutions will exist in many versions and refinements. A responsibility serves as a handle for discussing potential solutions. The responsibilities of an object are expressed by a handful of short verb phrases, each containing an active verb. The more that can be expressed by these phrases, the more powerful and concise the design. Again, searching for just the right words is a valuable use of time while designing."

Perhaps even more so, discovering just the right words is valuable when capturing discussions about business policies, business intelligence, and business activities. This knowledge is the requisite fodder consumed during requirements analysis and solution design, and it establishes the context within which software solutions will be developed and evolved.

### **Diagrams vs. Narratives**

How should knowledge about business problems be captured? What kind of residue should remain and be maintained after solution development? These questions often arise during software development projects. Most modern human interfaces are graphical, and so require graphical designs. Also, since the early 1980s, graphical notations have become a popular way to visualize other aspects of software.

Some methodologists have offered elaborate graphical notations for capturing knowledge about business elements and activities, e.g., the UML.<sup>2</sup> Such diagrams provide useful overviews and help us visualize the structural relationships between software elements, components, and composites. However, as design artifacts and residues, diagrams are expensive to create and maintain and usually require special tools. Also, they are prone to complexity and (intentionally) limited in their expressiveness.

Because of their rich structure and interconnections, most goals and usage scenarios are best expressed as structured narratives. Narratives are easy to create and maintain and usually require only simple tools. Also, narratives are easier to simplify and provide the full expressiveness of natural language. Given that modern software extensively leverages language metaphors, structured narratives offer a better fit for maintaining the traceability and the expressive proximity of the resultant software to its originating requirements.

### **Planning and Tracking Success**

Success is rarely accidental. It must usually be planned. This is especially true for software development. Software is a conceptual artifact. Software developers need conceptual leverage, techniques that make tackling the inherent complexity of software solution design tractable. First and foremost, software developers need to ensure that their work products align with the policies and improve the activities of a business.

Software solutions support and improve business activities. Business activities are directed by business policies. Business intelligence provides the management community with data critical to the guidance of business activities and the maintenance of their alignment with business policies.










Trace cards extend conceptual thinking beyond objects into their business origins. They provide ways to capture and share knowledge about business policies (vision and mission), business intelligence (qualities and quantities), and business activities (workflows and solution usage). They also help establish and maintain alignment and traceability across requirements levels.

Solution designs are usually impacted by various forces and constraints outside of the control of a solution designer. Engineers evaluate and attempt to balance the known forces based on their known or discovered effects. To properly evaluate and balance a set of forces, an engineer needs to know not only what these forces are, but also how they relate to each other. Software designers need to know not only *what* to accomplish and *how*, but also *why*. Trace cards provide answers to some of these questions.

## Goal Levels

Explicitly identifying the nature and level of goals can help us to align technology development with business objectives. In *Writing Effective Use Cases*,<sup>3</sup> Alistair Cockburn identifies several goal levels used for requirements. However, there are certain kinds of business policies that were not included. The following proposed goal levels extend those he offered and continues the metaphor of verticality he used.

**Table 1. Goal Levels**

Level	Nature	Source	Icon
Vision	Policy	Governor	
Mission	Critical Policy	Governor	
Mission	Central Policy	Governor	
Mission	Peripheral Policy	Governor	
Organization	External	Customer, Partner, Supplier	
Organization	Internal	Requestor	
System	External	Expector	
System	Internal	Interface Designer	
Component	External	Component Designer	

## Policy Cards

Business policies establish the ends, means, and central values of a business. These policies traditionally appear in business vision, mission, and value statements. These core business documents originate in the corporate governance community, especially the board of directors, the corporate officers, and the management team.

Policy cards generally derive their focus from business governance policy statements: vision, mission, values. Policy cards provide a way to capture and organize selected business policies on index cards. They can be used:

- to align software solution requirements with business policies,
- to trace software solution purposes back to business policies,
- as touchstones for prioritizing software solution features,
- throughout the various requirements levels.

A policy card should always capture the following information:

- A *quality instantiation* is a simple noun phrase composed of two parts:
  1. a descriptive adjective derived from the quality / value theme, and
  2. a common noun phrase that identifies a subject of concern under the identified theme.
- A *benefit description* describes the benefit(s) conferred by the instantiation.

A policy card may also exhibit one of two organizational variations: organized by theme and organized by subject. Card templates (1 and 2) for these two variations can be found in Appendix A.

When policy concerns are organized by theme (see Card Template 1),

- A *quality / value theme* abstracts a thematic concern from a vision, mission, or value statement.
- A *basic measure* characterizes the primary unit(s) of measure used to quantify the theme.
- The quality instantiations all share the same theme, but may have different subjects of concern.


When policy concerns are organized by subject (see Card Template 2),

- A *concern subject type* identifies a kind of subject: a process, a service, a product, or a policy.
- A *specific concern subject* identifies a specific subject of concern within the broader class.
- The qualify instantiations all share the same subject of concern, but may have different quality themes.

Policy cards can be used to capture ends statements, especially from vision and mission statements. However, ends statements differ from means statements in a subtle but significant way. Ends statements focus on effects rather than efforts. For some further remarks about ends statements and their formulation, see *Quality Alignment and Quality Inventories*.<sup>4</sup>


Here's a sample of how an overall corporate vision statement might be captured for Geekcorps (www.geekcorps.org).

### Card 1. Geekcorps Vision

<b>professionalism</b>	<b>engagement count</b>	
<i>professional</i> software practices	shared by qualified volunteer software professionals, and adopted by emergent organizations throughout the world, resulting in	
<i>better</i> software solutions	for emergent organizations worldwide	

Software solutions often have several associated service quality concerns, both on their surfaces and within their internals. While the surface qualities will impact users, IT (ops) will likely care about all of these qualities. For example, an online accident claim filing service might have the following external and internal quality concerns:

### Card 2. Claim Submission Service Qualities

<b>service</b>	<b>accident claim submission</b>	
<i>affordable</i> service	minimizes usage costs (user friendly)	
<i>available</i> service	maximizes opportunities for use (locations v. time)	
<i>quick</i> service	minimizes time to obtain result(s)	
<i>robust</i> service	handles exceptions gracefully (failure dialogs)	
<i>secure</i> service	prevents unauthorized usage (user authentication)	
<i>trustable</i> service	protects information privacy + integrity (SSL)	
<i>verifiable</i> service	provides evidence of commission (claim number, records, logs)	
<i>reliable</i> service	assures proper function	
<i>measurable</i> service	exposes relevant internal states and metrics	
<i>scalable</i> service	handles increased usage gracefully	

## Intelligence Cards

Business intelligence offers management the information needed to guide the business in the directions established by business policies. Business intelligence regards the various business quality concerns, especially the (quantitative) measurement of the relevant business products and activities.

Intelligence cards are generally paired: a quality card with a quantity card. A quality card collects knowledge about a single business quality concern, while the corresponding quantity card collects details about the measurements related to that quality concern. Together, they can be used:

- to capture the results of the goal, question, metric (GQM) approach,<sup>5, 6</sup>
- to align and trace business instrumentation requirements to business policies,
- throughout the various requirements levels.

Templates (3 and 4) for these two cards can be found in Appendix A. To link the quality and quantity cards, they both share the following information:

- A *quality concern* identifies a general quality area or issue of interest to a kind of stakeholder.
- A *quality concern subject* identifies a concern subject that needs assessment or improvement.
- An *essential metric* is used to measure (quantify) the quality of the concern subject.
- A *stakeholder role* identifies a kind of stakeholder concern about the focus quality.

A quality card (see Card Template 3) captures the following additional information:

- A *stakeholder objective* indicates the intended quality objective of the identified stakeholder.
- A *stakeholder goal / target* indicates a specific target and/or goal of the identified stakeholder.
- Each listed *question* helps characterize the concern subject from some stakeholder viewpoint(s).
- Each listed *metric* identifies the kind of measurement(s) needed to answer a listed question.

A quantity card (see Card Template 4) captures the following additional information:

- Each listed *measurement* identifies a characteristic needed to answer a concern question.
- Each listed *quality condition* identifies a relevant relation between measurements.
- Each listed *source / instrument* indicates the origin of a listed measurement or derivation of a listed condition.

Here's an example taken from the GQM literature.<sup>5</sup> We can detail the qualities and quantities needed to characterize and measure change request processing speed from the perspective of a project manager.

### Card 3. CR Processing Speed Questions

speed	cycle time
change request (CR) processing	project manager
minimize CR processing cycle time	10% faster by Q4
what is the current CR processing speed?	average cycle time standard deviation (from average)
what is the acceptable maximum cycle time?	<i>acceptable</i> limit <i>excessive</i> cases (over limit) percentage of <i>excessive</i> cases
is CR processing performance improving?	baseline, current average cycle time $( \text{current} / \text{baseline} ) * 100$ subjective satisfaction rating
is the CR process actually performed?	percentage of exceptions (review discoveries) subjective evaluation


### Card 4. CR Processing Speed Measures and Conditions

speed	cycle time
change request processing	project manager
<i>excessive</i> case percentage	$= ( \text{excessive cases} / \text{total cases} ) * 100$
<i>excessive</i> case	$= ( \text{request cycle time} > \text{acceptable limit} )$
<i>acceptable</i> cycle time limit	from: project manager
speedup percentage	$= ( \text{current average} / \text{baseline average} ) * 100$
average cycle time	$= \text{sum}( \text{case cycle time} ) / \text{total cases}$
request cycle time	$= ( \text{completion timestamp} - \text{initiation timestamp} )$
request completion timestamp	from: change request management system
request initiation timestamp	from: change request management system
process conformance evaluation	from: project manager (subjective)
performance satisfaction rating	from: project manager (subjective)
process exception percentage	$= ( \text{project exceptions} / \text{total cases} ) * 100$
process exception	from: project review (SEPG)


The following sample user story was taken from *Extreme Programming Installed*<sup>7</sup> (pg. 27).

"When the GPS has contact with two or fewer satellites for more than 60 seconds, it should display the message "Poor satellite contact," and wait for confirmation from the user. If contact improves before confirmation, clear the message automatically."

### Card 5. GPS Position Sense Stability Questions

<b>stability</b>	satellite signal strength	
GPS receiver position sense	GPS user	
maintain stable position sense	outages > 60 seconds acknowledged	
how strong is each satellite signal?	satellite signal strength (dB/mW)	
	<i>stable</i> satellite signal duration (seconds)	
how many satellite contacts are stable?	<i>stable</i> satellite signal duration >	
	<i>stable</i> satellite signal minimum (seconds)	
	<i>contacted</i> satellite count	
how long has the position been unstable?	<i>unstable</i> position sense duration >	
	<i>poor</i> satellite contact minimum (seconds)	

### Card 6. GPS Position Sense Stability Measures and Conditions

<b>stability</b>	satellite signal strength	
GPS receiver position sense	GPS user	
<i>poor</i> satellite contact	= ( <i>unstable</i> sense duration > <i>poor</i> contact minimum )	
<i>poor</i> satellite contact minimum	= 60 seconds (feature specifications)	
<i>unstable</i> position sense	= ( <i>contacted</i> satellite count < <i>stable</i> sense minimum )	
<i>stable</i> position sense	= ( <i>contacted</i> satellite count >= <i>stable</i> sense minimum )	
<i>stable</i> position sense minimum	= 3 contacted satellites (feature specifications)	
<i>contacted</i> satellite	when: receiver has a <i>stable</i> satellite signal	
<i>stable</i> satellite signal	= ( <i>strong</i> signal duration > <i>stable</i> signal minimum )	
<i>stable</i> satellite signal minimum	= 2 seconds (feature specifications)	
<i>strong</i> satellite signal	= ( signal strength > <i>strong</i> signal minimum )	
<i>strong</i> satellite signal minimum	= 20 dB/mW (feature specifications)	
satellite signal strength	from: receiver satellite signal strength meter	



## Face Cards

Face cards derive their name from their dominant focus on interfaces and interactions, the contracts and conversations that take place between interaction participants. Face cards provide a way to capture business activities and their stakeholders as use cases on index cards. A template for face cards can be found in Appendix A (see Card Template 5).

Face cards can be used as a starting point for writing use cases, or subsequently for use case analysis. Detailed discussions of use cases and their components can be found in *Writing Effective Use Cases*<sup>3</sup> by Alistair Cockburn. Face cards intentionally simplify use cases by eliminating some of the details that can be found in "fully dressed" use cases.

A face card captures at least the following information:

- A *primary goal* describes the completion condition expected by the *primary actor*.
- Each *stakeholder interest* describes a condition desired by the associated *stakeholder*.
- A *scenario trigger* describes a condition that the *trigger actor* effects to initiate the use case.
- Each *scenario step* describes an action of the associated *step actor*.

A face card may also capture preconditions and postconditions:

- Each *precondition* describes a condition that must be ensured by the associated *guarantor*.
- Each *minimal guarantee* describes a postcondition that will always be ensured for a *recipient*.
- Each *success guarantee* describes a success condition that will be ensured for the *recipient(s)*.

The format of a face card separates stakeholders (in the left column) from their interests, expectations, obligations, and responsibilities (in the right column). Most situations are expressed using an active verb phrase with present tense. Two exceptions include preconditions and some triggers. Preconditions use an active verb with past tense. Some triggers may also use past tense.


As with use cases in general, face cards can be used to capture interactions for several goal levels. When used in conjunction with other trace cards, they are especially appropriate for the following three:

Goal Level	Focus	Potential Inclusions
business interface	business customers + partners, interests, expectations, contractual obligations, collaborative responsibilities	exchanges of: value, goods, information
human interface	users + business stakeholders, interests, expectations, obligations, collaborative responsibilities	information exchanges
component interface	collaborators + clients, quality expectations + obligations, functional responsibilities	information exchanges

When used at the later, more detailed requirements levels, face cards can serve as an intermediate residue between use cases and design. They can help identify candidate object classes and responsibilities at human and component interfaces. This knowledge can then be fed forward into responsibility-driven object-oriented design, especially with CRC cards.

The following two face cards were derived from the use case named "Get Paid for Car Accident" in *Writing Effective Use Cases*<sup>3</sup> (pg. 5). The first face card shows the stakeholders with their interests, the postconditions, and the main success scenario, including the scenario initiation *trigger*.

### Card 7. Accident Payment Main Success Scenario

<b>claimant</b>	<b>gets paid for a car accident claim</b>	
claimant	wants to be paid the most possible	
insurance company	wants to pay the smallest appropriate amount	
state insurance department	wants to see that all guidelines are followed	
insurance company	(always) logs the claim and all activities	
claimant + insurance co	agree on the amount to be paid	
claimant	gets paid the agreed amount	
claimant	<i>submits a claim with substantiating data</i>	
insurance company	verifies that claimant owns a valid policy	
"	assigns an agent to examine the claim	
"	verifies that all details are within policy guidelines	
"	pays claimant	
"	closes file	

The second face card shows the extension scenarios.

### Card 8. Accident Payment Extension Scenarios

<b>claimant</b>	<b>gets paid for a car accident claim</b>
claimant	<i>submitted an incomplete claim</i>
insurance company	requests missing information
claimant	supplies missing information
claimant	<i>did not own a valid policy</i>
insurance company	denies claim, notifies claimant
"	records all this, terminates proceedings
claimed accident	<i>violated basic policy guidelines</i>
insurance company	denies claim, notifies claimant
"	records all this, terminates proceedings
claimed accident	<i>violated some minor policy guidelines</i>
insurance company	negotiates a settlement amount with the claimant

## Further Considerations

From a certain perspective, the prevailing state of software development practice has inverted priorities. Typically, we rush (or are rushed) to implement solutions before we sufficiently understand the problems we're being asked to solve. Stakeholders tacitly assume that software solutions will improve the quality of their lives and business activities. But, such benefits still elude our collective grasp far too often. Without giving quality due consideration, without spending significant time discussing quality expectations explicitly and systematically, the achievement of *satisficing* solutions will continue to be accidental and haphazard rather than intentional and sure. The "rubber meets the road" where requirements meet designs.

Historically, the practice of responsibility-driven design with CRC cards has focused primarily on knowledge (noun phrases) and behavior (verb phrases) to the virtual exclusion of qualities (as expressed by descriptive adjectives). The absence of quality responsibilities can usually be traced back to the absence of quality considerations in the original requirements. Quality requirements often receive little or no consideration during requirements gathering, especially in comparison with the emphasis often placed on form and function.

But, the consideration of object responsibilities should properly include qualities in addition to knowledge and behavior. Conceptually, we can understand qualities as expressive of the dimension (and states) of being. So, objects have and *ensure* qualities in addition to *having* knowledge and *doing* behaviors. Objects should know *why* in addition to *what* and *how*. Every object

<i>knows</i>	some information (with structure)	know <i>what</i>
<i>does</i>	some behaviors (with functions)	know <i>how</i>
<i>ensures</i>	some qualities (with conditions)	know <i>why</i>
<i>is</i>	some concept (with a role and relations)	

In practice, object-oriented designs entail strong contracts between servers and their clients. So, a server object imposes responsibilities (especially for quality) on its clients as preconditions to the proper usage of its services. Likewise, a server object offers quality guarantees to its clients (as postconditions and invariants). Method preconditions (ensured by clients), method postconditions (ensured by servers), invariants (ensured by servers), constraints, and qualified class names all offer the potential for consideration with respect to quality requirements.

For example, reconsider Card 6 above. This example demonstrates how user stories can be analyzed for their goals, and how apparently simple goals can entail quite elaborate quality and measurement requirements when explored in detail. Notice that the definitions of named quality conditions often relate values, e.g., from measurements, like the following:

*stable* position sense =  
 ( *contacted* satellite count  $\geq$  *stable* position sense minimum )

During requirements analysis (and during design), it's often useful to explore how quality themes and conditions decompose into value relations and the measurements and metrics needed to test them. Explicitly identifying and naming test conditions also improves the testability of object-oriented designs. Intelligence cards can be used to capture and share the results of such analysis. Thus, identifying and naming quality conditions and their component value relations can play an important part in ensuring software solution quality.

Perhaps this extended understanding of responsibilities can help improve our object-oriented designs, especially in the context of CRC cards. Both client and server responsibilities can be listed on a CRC card. Enumerating only the responsibilities for knowledge and behavior weakens a class design, while also enumerating the respective responsibilities of both a server and its clients strengthens a design into a contract, especially when quality responsibilities are included. This simple practice also increases the testability of the design if the quality conditions are ultimately exposed as test methods in the implementation. So, this practice can serve as a natural adjunct for test-driven development approaches like extreme programming (XP).<sup>8</sup>

**Table 2. Card and Traceability Summary**

<b>Card Type</b>	<b>Knowledge</b>	<b>Sources</b>	<b>Traceability Intent</b>
Policy Card	quality concerns, benefits (value)	governors: board + officers, vision, mission, values	ties to business policy statements: vision, mission, values
Intelligence Cards	concerns, goals, metrics, measurement linkage	requestors: activity managers, other stakeholders, objectives, goals	ties business intelligence requirements back to business policies and priorities
Face Card	goals, interests, expectations, obligations, interactions	expectors: activity participants, users and other stakeholders, business contracts, user stories	ties business activities and activity improvement goals back to business policies and priorities
CRC Card	responsibilities for: quality, behavior, knowledge	developers: design sessions and decisions	ties solution designs back to business activities and business intelligence requirements

## References

1. Kent Beck, Ward Cunningham. A Laboratory for Teaching Object-Oriented Thinking. *OOPSLA 1989 Conference Proceedings*. ACM SIGPLAN Notices 24(10), 1989.
2. Object Management Group. UML™ Resource Page. <http://www.omg.org/uml/>.
3. Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Publishing, Inc., 2000. ISBN 0-201-702258.
4. Nik Boyd. *Quality Alignment and Quality Inventories*. <http://www.educery.com/papers/rhetoric/quality/alignment.htm>.
5. Victor Basili, Gianluigi Caldiera, Dieter Rombach. The Goal Question Metric Paradigm. *Encyclopedia of Software Engineering*. John Wiley & Sons, Inc., 2001. ISBN 0-471-37737-6.
6. Scott Whitmire. *Object-Oriented Design Measurement*. John Wiley & Sons, Inc., 1997. ISBN 0-471-13417-1.
7. Ron Jeffries, Ann Anderson, Chet Hendrickson, Kent Beck. *Extreme Programming Installed*. Addison-Wesley Publishing, Inc., 2001. ISBN 0-201-70842-6.
8. Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Publishing, Inc., 2000. ISBN 0-201-61641-6.

## Appendix A. Trace Card Templates

Template 1. Theme-Oriented Policy Card

<b>quality / value theme</b>	<b>basic measure</b>
quality instantiation	benefit description

Template 2. Subject-Oriented Policy Card

<b>concern subject type</b>	<b>specific concern subject</b>
quality instantiation	benefit description



**Template 5. Face Card**

<b>primary actor / role</b>	<b>primary goal</b>
stakeholder	interest(s)
minimal guarantee recipient	minimal guarantee
success guarantee recipient	success guarantee
precondition guarantor	precondition
trigger actor	<i>scenario trigger</i>
step actor	scenario step (activity)