# Reducing the Effects of Requirements Changes through System Design

Israel Navarro, Nancy Leveson, Kristina Lundqvist
Massachusetts Institute of Technology
Software Engineering Research Laboratory, 33-313
Cambridge, MA 02139

## ABSTRACT

The continuous stream of requirements changes that often takes place during software development can create major problems in the development process. This paper defines a concept we call *semantic coupling* that along with features of intent specifications can be used during system design to reduce the impact of changing requirements. The practicality of using the approach on real software is demonstrated using the intent specification of the control software for a NASA robot designed to service the heat resistant tiles on the Space Shuttle.

## 1. INTRODUCTION

Requirements changes cause havoc in a development process when the effects of the changes ripple through a large part of the system and software design, which then affects other parts until the overall design and conceptual model may be affected and start to degrade. Hopefully, the highest-level, most abstract requirements will not change, but sometimes they do, as system requirements become better understood. Reversals in TCAS (an airborne collision avoidance system required on almost all commercial aircraft in the U.S.) are an example of this problem. About four years after the original TCAS requirements specification and system design were completed and TCAS development was at the point of being tested on commercial flights, experts discovered that the design did not adequately cover the case where the pilot of an intruder aircraft does not follow his or her TCAS advisory, and thus TCAS must alter the advisory to its own pilot. This change in basic requirements (the need to reverse advisories) caused extensive changes in the TCAS system and software design, some of which introduced additional subtle problems and errors that took years to discover and rectify.

While the highest—most abstract and global—level requirements, such as detecting potential threats in TCAS, are less likely to change than lower-level requirements, when they do they have the most important (and costly) repercussions. A change at the highest level (or at any level) may require changes and reanalysis at all the levels below it. The impact of change in safety-critical systems can be extremely costly as changes can require a new system safety analysis. Any reduction in the effort required to accomplish this analysis will have a tremendous impact on cost and schedule.



We believe the reason requirements changes cause so much disruption to a project is rooted in the concept of *semantic coupling*. In general, tightly coupled systems are highly interdependent: Each part is linked to many other parts, so that a failure or unplanned behavior in one can rapidly affect the others.

The concept of software module coupling is not new; it has been used in software design for over 20 years to make software components more composable and reusable and to enhance the understandability of individual modules so they are easier to modify. *Structural coupling* characterizes a module's relationship to other modules, that is, it measures the interdependence of two modules. The more connections between modules, the more dependent they are. Modularity and low coupling help confine the effects of changes to a single module.

Yourdon and Constantine [10] and Myers [7] identified three factors related to module coupling: the number of interfaces, the complexity of the interfaces, and the type of information flow (data or control) along the interfaces. These factors affect visibility outside the interface, the amount of information exported through interfaces, and the required interaction among programmers. Five types of module coupling have been identified, but in general modules are considered tightly coupled if they use shared variables or if they exchange control information. Software engineering attempts to define coupling have been limited to software module design and primarily to structural rather than semantic dependencies.

The counterpart to coupling is cohesion. A module could be said to have high cohesion if all the elements have a strong relationship to each other. Nine types of cohesion have been identified. Some of these types of cohesion are related to functionality, while others are not. All have been defined rather vaguely, usually using examples alone.

The need for these measures of the "goodness"' of a design stem from the more general problem of complexity or intellectual manageability of the systems we are trying to build. Intellectual manageability can be enhanced by organizing information in a coherent, structured form, by using appropriate abstractions, and by reducing the dependencies among system components [7]. At the code level, this approach led to the concepts of coupling and cohesion.

In this paper, we show how a new organization of information, called an intent specification, along with a concept we call semantic coupling, can assist in creating system designs that reduce the effects of requirements changes. Just as module coupling has been used to guide the structural decomposition of modules, semantic coupling could be used to guide the process of

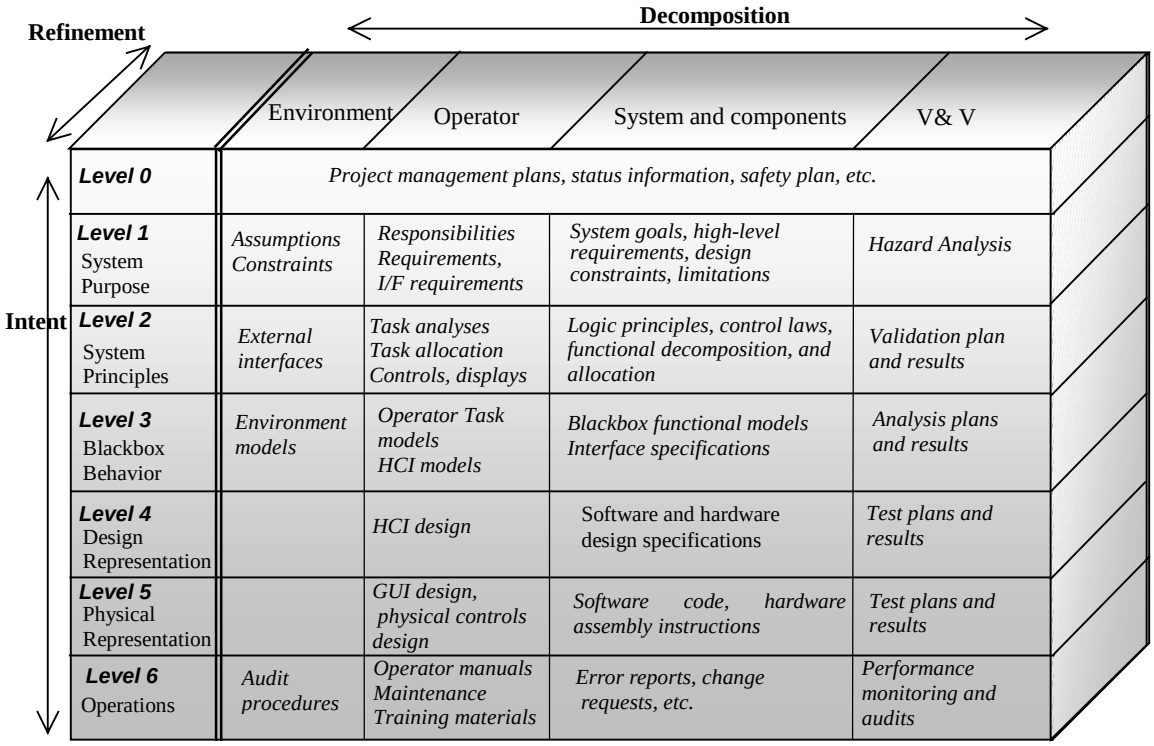



**Figure 1. The Structure of an Intent Specification.**

functional decomposition of requirements and high-level system design.

The next section presents the basic concepts of Intent Specifications. Then semantic coupling is defined and a process outlined for identifying and reducing it. To determine the feasibility of applying the concept to real systems, we have applied the process to the Mobility and Positioning Software (MAPS) of a robot designed for NASA to service the heat-resistant tiles on the Space Shuttle. The results of this study are described.

## 2. INTENT SPECIFICATIONS

The design of *intent specifications* applies research in systems theory, cognitive psychology, and human-machine interaction to specification design in order to support the tasks humans perform during system and software development [4]. The structure, content, and notation of intent specifications are designed to assist software engineering in developing appropriate mental models about the system while supporting them in using a wide array of problem-solving strategies.

Intent specifications employ a type of hierarchical abstraction based on intent or purpose. In computer science, we have made extensive use of hierarchical abstraction to allow both top-down and bottom-up reasoning about complex systems. Two types of abstraction have been used: (1) part-whole abstractions where each level of a hierarchy represents an aggregation of the components at a lower level and (2) information hiding (refinement) abstractions where higher levels contain the same conceptual information but hide some details about the concepts, that is, each level is a refinement of the information at a higher level.

Using these hierarchical abstractions, each level of the usual software specifications can be thought of as providing "what" information while the next lower level describes "how." Such hierarchies, however, do not provide information about "why." Information about purpose or intent (i.e., design rationale) cannot be inferred from what we normally include in such specifications. Design errors may result when we either guess incorrectly about intent or omit it from our decision-making process.

Similar problems are faced by cognitive engineers and human factors specialists in designing interfaces for operators of complex systems. The human-machine interface provides a representation of the state of the system that the operator can use to solve problems and perform tasks, such as monitoring and diagnosis. Software specifications can be thought of as an "interface" between the software and the maintainer of that software or between the designer and the implementer. Intent specifications apply these ideas in operator-machine interface design to the "interface" provided by system and software specifications.

Intent specifications are organized along three dimensions: part-whole, refinement, and intent (see Figure 1). The vertical dimension specifies the level of intent at which the problem is being considered, i.e., the language or model that is currently being used. The horizontal decomposition (part-whole) and refinement dimensions allow users to change their focus of attention to more or less detailed views within each level or model. The information at each level is fully linked to related information at the levels above and below it.

Each level of the intent abstraction contains information about goals or purpose for the level below, described using a different set of attributes or language. Higher level goals are not constructed by integrating information from lower levels; instead each level provides different, emergent information with respect

to the lower levels. A change of level represents both a shift in concepts and structure for the representation (and not just a refinement of them) as well as a change in the type of information used to characterize the state of the system at that level. Thus each level is a different view or model of the entire system and may use a different language.

Mappings between levels are potentially many-to-many: Components of the lower levels can serve several purposes while purposes at a higher level may be realized using several components of the lower-level model. These goal-oriented links between levels can be followed in either direction. Changes at higher levels will propagate downward, i.e., require changes in lower levels, while design errors at lower levels can only be explained through upward mappings (that is, in terms of the goals the design is expected to achieve). In this paper, we show how the structure of these mappings from one intent level to another define the semantic coupling between the components at the higher level and can be designed to reduce this coupling..

Intent specifications assist in identifying intent-related dependencies. Because each level is mapped to the appropriate parts of the intent levels above and below it, traceability of not only requirements but also design rationale and design decisions is provided from high-level system goals and constraints down to code (or physical form if the function is implemented in hardware) and vice versa. This mapping is not a mathematical relationship, but a mapping based on design rationale and intent. For example, while one level might describe the logic of TCAS II in terms of whether an intruder aircraft is on the ground or not, the next higher level might define "on the ground" (which is surprisingly complex) and why the engineers chose that particular definition [6].

Consideration of purpose or reason (top-down analysis in an intent hierarchy) has been shown to play a major role in understanding the operation of complex systems [8]. Experts and successful problem solvers tend to focus first on analyzing the functional structure of the problem at a high level of abstraction and then narrow their search for a solution by focusing on more concrete details [2]. Representations that constrain search in a way that is explicitly related to the purpose or intent for which the system is designed have been shown to be more effective than those that do not because they facilitate the type of goal-directed behavior exhibited by experts [9]. Therefore, intent specifications should be able to improve the problem solving required in software development and evolution tasks by providing a representation of the system that facilitates goal-oriented search.

Our current conception of intent specifications has seven hierarchical levels, each level supporting a different type of reasoning about the system. The Management level (Level 0) provides a bridge from the contractual obligations and the management planning needs to the high-level engineering design plans. The System Purpose (Level 1) assists system engineers in their reasoning about system-level goals, constraints, and limitations. It also documents basic hazard analysis and contains a hazard log to track the safety engineering activities. The system safety analysis is used to establish safety-related system and software requirements and design constraints.

The System Principles level (Level 2) documents the system in terms of the physical principles and laws upon which the design is based. Complete functional structures and interfaces between components are defined at this point. This level also documents task models and other results of human factors analyses for systems containing human operators. Display and control information is defined as well.

The Blackbox level (Level 3) allows the designers to study the logical design of the system using a formal modeling language called SpecTRM-RL [5]. The tools associated with this language produce executable and analyzable models that engineers can use to study the complex interactions between the different components of the system. SpecTRM-RL models can be built for the non-software components also, or they can be executed together with simulations and prototypes of hardware components.

The Design Representation level (Level 4) presents a detailed implementation-dependent software design as well as any applicable hardware design specifications. The Physical Representation level (Level 5) contains the actual software and hardware implementation of the system as well as any necessary training and maintenance manuals. Finally, the System Operations level (level 6) includes information produced during the actual operation of the system, which can be used in operational audits, performance analyses, operational safety analyses, and change analysis.

Each level also contains documentation of the plans for and results of the verification and validation process for the design decisions contained in each level.

## 3. SEMANTIC COUPLING

Intent specifications [4] allow isolating the assumptions that need to be considered or reconsidered to make a particular change and also provide traceability between requirements and design and each level of the intent hierarchy. Therefore, they may not only help in understanding the system and the relationships between requirements and design so that changes can be made correctly, but may also be helpful in validating any changes to make sure that they do not violate the intent of the original designers (and thus introduce serious errors in changing something important).

While recording design rationale in a usable and traceable way should reduce the effort involved in responding to changed requirements, it is still possible to design intent specifications for which changes in requirements will cause extensive changes (rippling effects) at each level of the specification. To assist with this problem, we introduce the concept of semantic coupling.

Semantic coupling or independence is defined in terms of the mappings between the levels of the intent specification, that is, mappings between the goals and the means for achieving the goals. The emphasis in this paper will be on the first two levels, but the same definition applies at each level. At the lower levels of an intent specification, where the logic is mapped to software structures and modules, the concept is identical to that of module cohesion and coupling, but the functional aspects of these concepts can be more carefully defined by tracing the functions back up the levels of the intent specification (the functional decomposition process).

Using these mappings, three levels of coupling or independence can be identified:

1. Uncoupled: the mappings or functions from requirements to design principles are all one-to-one.
2. Loosely coupled: the mappings are one-to-many.
3. Tightly coupled: the mappings are many-to-many.

For any complex control system, a completely uncoupled design, while allowing changes to requirements with minimal impact, is usually not practical. A more realistic goal is to design to reduce the impact of requirements changes, i.e., to eliminate the rippling effects to other requirements and therefore through the system design as a whole by designing the mappings to be one-to-many. Our design goal then becomes: eliminate (or if not possible, reduce) many-to-many mappings and create a design such that the one-to-many mappings are reduced. The latter can be accomplished by a careful ordering of design decisions and hierarchical ordering of requirements, as illustrated below. For ease of manipulation and visualization, we describe the functions using Matrix Algebra constructs. Thus, the relationship between the requirements (R) and the system design principles (SDPs) can be characterized mathematically by expressing them in terms of vectors:

(1) {FR} = [A] {SDP}

where {R} is a vector of m requirements, {SDP} is a vector of n system design principles, and [A] is the traceability matrix. The values of the traceability matrix will be either * or 0, where * indicates a mapping exists between the corresponding vector components while 0 signifies no mapping.

Consider the case of a square traceability matrix where m=n=3. The simplest and most desirable design, an *uncoupled* design, is one in which all the non-diagonal elements of the traceability matrix are zero. That is:

$$(2)\quad [A] = \begin{bmatrix} A_{11} & A_{21} & A_{22} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{bmatrix}$$

In the uncoupled design, each of the FRs can be satisfied independently from the others. If a requirement (R) changes, all one needs to do is to adjust the corresponding system design principle (SDP). Note that in software, the traceability matrix will almost always not be square, that is, there will be multiple SDPs for each requirement.

A *coupled* design is one in which elements on both sides of the diagonal are non-zero, as in:

$$(4)\quad \begin{pmatrix} FR_1 \\ FR_2 \\ FR_3 \end{pmatrix} = \begin{bmatrix} * & * & 0 \\ 0 & * & * \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_1 \\ SDP_2 \\ SDP_3 \end{pmatrix}$$

In this case, a change in any of the requirements is likely to produce a cascading effect, forcing all three SDPs to be adjusted. In general, any requirements changes in a coupled design can require a highly complex, iterative redesign of the system. This process can be extremely expensive, particularly if the changes take place far into the detailed design or implementation phase.

The third case is the *loosely coupled* design, where the traceability matrix is lower or upper triangular:

$$(5)\quad \begin{pmatrix} FR_1 \\ FR_2 \\ FR_3 \end{pmatrix} = \begin{bmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_1 \\ SDP_2 \\ SDP_3 \end{pmatrix}$$

In a loosely coupled design, even though the requirements are not strictly independent, the impact of potential changes can be reduced by adjusting the SDPs in the proper order. Thus, as stated above, while total independence or uncoupling is almost always not achievable for real systems, the effects of changes in a loosely coupled design can be reduced by a careful ordering of design decisions and hierarchical ordering of requirements. Our goal then, is to create designs that are loosely coupled and to order the design decisions such that changes will have minimal impact.

The process above is oversimplified because different types of requirements will affect the design in different ways. In an intent specification, Level 1 contains three types of "requirements": functional requirements[1], environmental assumptions, and design constraints.

| | | System Design Principles | | |
|---|---|---|---|---|
| | | SDP1 | SDP2 | SDP3 |
| MAPS High-level FRs | | | | |
| FR1 | | ■ | | |
| FR2 | | | ■ | |
| Environmental Assumptions | | | | |
| EA1 | | ■ | ■ | |
| EA2 | | | | ■ |
| Design Constraints | | | | |
| C1 | | | ■ | |
| SC1 | | ■ | | |

**Figure 2. Example Traceability Matrix.**

Figure 2 shows the general form and simple example of a traceability matrix. The three types of requirements are separated as they involve tradeoffs with each other or have different implications for system design.

Functional requirements are derived from general system goals, i.e., the mission of the system. The top part of Figure 2 shows an example of high-level functional requirements (FR1 and FR2) and their mapping to the system design principles (SDP1 and SDP2) that is clearly decoupled: SDP1 and SDP2 independently satisfy FR1 and FR2, respectively.

Environmental assumptions lead to a second type of requirement that arises as soon as the system boundary is defined and the system interface thus becomes bound to (dependent upon) assumptions about the components in the environment. Intent specifications include specification of environmental assumptions

[1] We do not separate performance requirements from functional requirements. In a real-time system, this separation is not possible and when attempted, as often occurs, leads to many of the problems that arise in building such systems. The reason behind this claim is beyond the scope of this paper. See Jaffe [3] for an explanation.

at Level 1 and the mapping of these assumptions to Level 2 system design principles. In Figure 2, the links connecting the environment assumptions and system design principles indicate that those SDPs depend on assumptions about the behavior of components in the system environment (the system interface). This dependency can be a simple data dependency (format, range, or time properties) but can also include behavioral information (the external system is expected to accomplish certain tasks, etc.). Detailing these data dependencies is important in defining a sound system interface. In addition, some environmental dependencies can lead to undesirable semantic couplings. In Figure 2, both SDP1 and SDP2 are dependent on environmental assumption EA1 and thus this system is coupled with respect to the environmental assumptions; changes in this assumption could have a large impact on the design. Appropriate design changes or tradeoff decisions should take this coupling into account in order to reduce its effects.

The third type of requirement, design constraints (DCs), specify restrictions on the way the system can satisfy the requirements. Constraints may be safety constraints, e.g., the operation of TCAS must not interfere with the ground-based air traffic control system, or non-safety constraints, e.g., TCAS-generated avoidance maneuvers must minimize the impact of the evasive maneuver on the aircraft flight path. Safety constraints are derived from the system hazard analysis performed for safety-critical systems. Other design constraints (i.e., not safety related) may be imposed by the customer or potential users of the system or may result from non-mission related requirements imposed on the system design.

The mappings from design constraints (DCs) to SDPs have a different meaning than those linking FRs and SDPs. The SDPs must not violate the DCs, but they also do not necessarily implement a safety function nor does the function necessarily need to be modified to accommodate the constraint. Thus, two SDPs are not coupled just because the same safety constraint applies to them. However, any change to a SDP linked to a DC means that an analysis must be triggered if any changes are made in the SDP in order to ensure that safety has not been compromised. Thus constraints need not be independent from the FRs and they do not necessarily couple DCs, but the way in which the SDPs limit system functionality in order to satisfy any applicable constraint could also affect the ease and cost associated with system changes and evolution.

Figure 2 also shows a SDP linked to an EA but no FR, i.e., the link between SDP3 and EA2. This situation can occur when a level of the intent specification adds design features or design decisions that are useful or necessary at that level to complete the design or model but are not part of the functional requirements. These SDPS may be related to environmental assumptions or to design constraints or may simply be added by the designers for other reasons.

## 4. CASE STUDY: MAPS

The software chosen for the case study is the Mobility and Positioning Software (MAPS), part of the control software for a CMU/NASA robot [1]. The robot was designed to inspect and waterproof between flights each of the 17,000 silica tiles that protect the Space Shuttle underside. Because there are so many tiles, the robot divides its work area into uniform work spaces, inspecting tiles in each area with as little overlap between work spaces as possible.

Before each inspection shift, a supervisor enters instructions and information about shuttle position and inspection sequence via an off-board computer, the Workcell Controller. The Workcell Controller workstation is used to create jobs and update other NASA databases after the robot uploads data gathered during the course of the shift. This data includes tile images, records of tiles injected and inspected, and other pertinent job data. In addition, robot status data is used to monitor robot operation.

At the beginning of the shift, a job is downloaded to the robot. A job consists of a series of files describing the locations, sequences, target IDs, orbiter parking measurements, etc. The robot then uses a rotating laser to position itself under the shuttle, and the robot's camera locates the exact tile to be inspected. Because the shuttle's belly is not flat, the robot must customize its upward movement to each tile: Two vertical beams on either side of the robot raise the manipulator arm, which holds the injection tools and camera. A smaller lifting device raises the arm the rest of the way. By comparing the current state of each tile with the state of the tile at previous inspections, the robot characterizes anomalies in tiles as cracks, scratches, gouges, discoloring, or erosion. The robot also indicates when it is unsure what is wrong with a tile, so the supervisor can reanalyze the tile on the screen of the Workcell Controller. At the end of a shift, the robot's updated tile information is entered into existing NASA databases.

On board the robot, a computer controls the robot's high-level processing tasks while a low-level controller and amplifiers direct arm and wheel motions. Two more computers control the robot's vision and injection systems. If anything goes wrong, such as rising compartment temperatures or low battery level, safety circuits will shut down the robot.

MAPS is in charge of issuing low-level commands to the Motor Controller based on the inputs received either from the Planner (AI-based software) or a human operator. The Planner controls robot movement and positioning by providing MAPS with a specification of the destination and route. The operator controls robot movement and positioning using a hand-held joystick. The robot is unstable when the manipulator arm is extended, so stabilizer legs are used to provide stability. These legs must be retracted when the robot is in motion. MAPS is responsible for controlling the stabilizer legs. MAPS also monitors and controls several other robot subsystems and is responsible for most safety-related functions.

Level 1 of the MAPS intent specification contains the system goals, high-level functional requirements, limitations, constraints, and hazard analysis. It also includes the assumptions MAPS makes about the other robot subsystems and relevant external factors. Level 2 contains the MAPS design principles. While we have completed the top three levels of the MAPS intent specification as well as the traceability matrices, they are too large to include here. Instead, we describe in detail two parts of the specification to illustrate how the concept of semantic coupling can be applied to a real system. The information is the actual system requirements and design principles have been simplified for space and conciseness purposes. Downward links are also omitted as they are not relevant for the examples.

## 4.1 MAPS High-level Functionality

The first example focuses on the definition of the different modes under which MAPS can function. Following an initial assessment of the system goals and safety concerns, the designers identified the need to make the robot controllable both manually or automatically. These requirements originate in the need to provide a balance between the performance goals and the safety constraints. The FRs that correspond to these requirements follow:

**MAPS Level 1 High-level Functional Requirements**

**MAPS-1.1.1: Computer-Controlled Operations**

MAPS shall be able to operate automatically through control of the Planner alone. *Rationale: Human control of MAPS throughout the long and tedious tile servicing process (which takes several weeks) is impractical.*

**MAPS-1.1.2: Operator-Controlled Operations**

MAPS shall be able to be operated under direct manual control. *Rationale: Sufficient confidence cannot be obtained in the automated implementation of some safety-related robot operations (like detecting a passerby or an unexpected obstacle in the path of the robot), and therefore human movement control will be used during some limited but particularly hazardous operations.*

These FRs result in the definition of a series of modes and associated transitions in level 2 of the intent specification. However, these FRs are not adequate to produce a complete design. Assumptions about the robot's other components and about the way in which MAPS is supposed to interact with them must be known. For this example, the engineers need some basic assumptions about the Planner (PL), the Motor Controller (MC), and the operator joystick interface (JOY):

**MAPS Level 1 Environmental Assumptions**

**PL1.2:** The Planner will provide both a route of travel and a destination to MAPS in world coordinates.

**MC3:** The Motor Controller can be operated in position (relative displacement) mode.

**MC4:** The Motor Controller can be operated in velocity mode.

**MC6:** The Motor Controller is able to stop the motion of the robot within 0.2 seconds of receiving a 'stop' command.

**JOY1:** The operator will be able to drive the robot by deflecting a joystick in the direction the operator would like the robot to travel

Level 1 of the intent specification also includes a preliminary hazard analysis from which a set of safety constraints (SC) is derived. The SCs limit the way in which the functionality required by the FRs can be implemented. Two SCs related to the example are:

**MAPS Level 1 Safety Constraints**

**SC1:** The robot mobile base must move only when commanded.

**SC2:** The robot mobile base must stop when commanded.

During the design process, system engineers must define a functional structure out of all the Level 1information. This synthesis process is quite challenging; it requires from the designers a multi-disciplinary approach where the optimization of functional requirements may conflict with the safety constraints.

The level 2 Design *Principles* corresponding to the current example are:

**MAPS Level 2 Design Principles**

**MAPS-2.2.1: Initialization Precondition**

MAPS does not accept any motion commands until system initialization mode is complete [↑SC1]. *Rationale: The normal operation of the system relies on the correct performance of the subsystems initialized during startup.*

**MAPS-2.2.2: Control Mode Selection Principles**

At any time, MAPS is in one and only one of the following three modes: safety mode, computer mode, or joystick mode. Mode selection is based on the following principles: […]. [↑MAPS 1.1, ↑MAPS 1.1.1, ↑MAPS 1.1.2,↑SC2]

**MAPS-2.2.3: Safety Mode**

In Safety Mode, MAPS stops the robot, prevents further motions, notifies the operator of the problem that has occurred, and executes a set of recovery procedures. [↑SC2, ↑MC6] *Rationale: This mode provides MAPS with the ability to handle unsafe conditions and to return the robot to a safe state if possible.*

**MAPS-2.2.4: Computer-Controlled Operations**

When operating in Computer mode, MAPS accepts routes from the Planner and generates all the necessary movement commands for the motor controller to move the robot along that route. Relative displacement mode is used. Motor controller commands are generated by calculating […] [↑MAPS-1.1.1, ↑MAPS-1.1.1.1, ↑MC3, ↑PL1.2, ↑SC1] *Rationale: This mode permits MAPS to position the robot very precisely and allows the optimization of the waterproofing operations.*

**MAPS-2.2.5: Operator-Controlled Operations**

When operating in joystick mode, MAPS accepts movement commands from the operator via the joystick and issues all necessary motor controller commands to move the robot as commanded by the operator. Velocity mode is used when operating in joystick mode. In velocity mode, the kinematics on the body relative velocity are computed, and the appropriate wheel velocities are set. The computations used for the kinematics are […] [↑MAPS-1.1.2, ↑MC4, ↑Joystick1, ↑SC1] *Rationale: Velocity commands best match an operator's mental model of how robot base motions are controlled, i.e., they are the easiest for humans to understand and therefore monitor.*

Note how most of the level 2 design principles include the rationale behind design decisions as well as traceability information. This information can prove extremely useful in system redesigns and during maintenance by providing insight into the reasoning and assumptions of the original designers. The traceability links are denoted by up-arrows (↑), providing an explicit way to indicate a dependency or satisfying relationship between different items of an intent specification. Our automated tools to support intent specifications use hyperlinks.

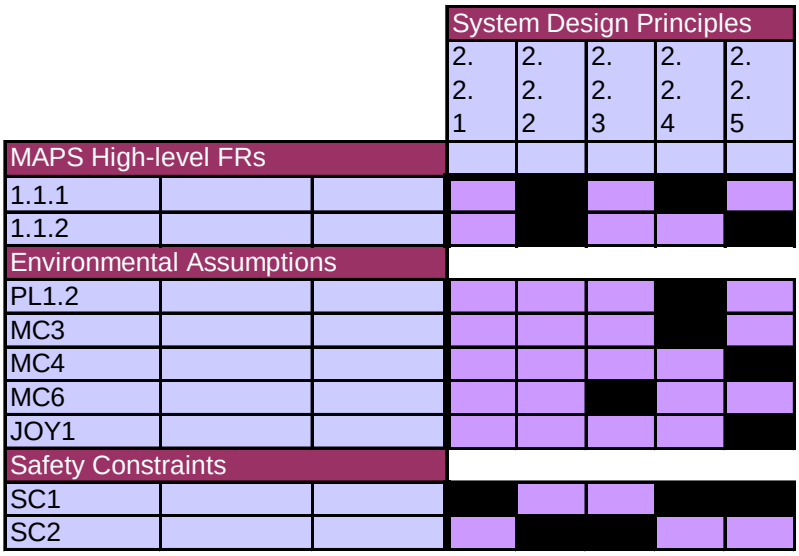



**Figure 3. MAPS Traceability Matrix.**

Creation of the traceability matrix for this example is immediate from the links:

$$\begin{pmatrix} FR_{1.1.1} \\ FR_{1.1.2} \end{pmatrix} = \begin{bmatrix} * & * & 0 \\ * & 0 & * \end{bmatrix} \begin{pmatrix} SDP_{2.2.2} \\ SDP_{2.2.4} \\ SDP_{2.2.5} \end{pmatrix}$$

(6)

As expected, the traceability matrix is not square. Equation (6) provides important information regarding the coupling in the system. $FR_{1.1.1}$ is only satisfied by $DP_{2.2.2}$ and $DP_{2.2.4}$. Similarly, $FR_{1.1.2}$ is satisfied by $DP_{2.2.2}$ and $DP_{2.2.5}$. This means that the implementation of each type of operation is accomplished by separate control modes, each independent of each other. The only DP that is common to both functional requirements is the one defining the mode selection logic. This is clearly a case of a loosely coupled design. This point can be seen easily bu expanding the matrix:

$$FR_{1.1.1} = SDP_{2.2.2} + SDP_{2.2.4}$$

$$FR_{1.1.2} = SDP_{2.2.2} + SDP_{2.2.5}$$

(7)

If the designer can fix $DP_{2.2.2}$ before the other DPs, then $DP_{2.2.4}$ and $DP_{2.2.5}$ can be adjusted independently to satisfy their corresponding functional requirements. In practice, this means that the mode transitions should be defined apart from the mode functionality itself. In addition, the implementation of each mode must be unaware of the existence of other modes. In this way, if one of the two modes needs to be changed or removed altogether, the remaining mode can be left unchanged. The traceability matrices can be used to identify subtle interactions and dependencies between the modes.

The environment assumptions and safety constraints affecting the DPs are shown in Figure 3. Note how the environmental assumptions corresponding to $SDP_{2.2.4}$ and $SDP_{2.2.5}$ were successfully encapsulated: they do not have any common assumptions.

Figure 3 also contains DPs that are not linked to any level 1 FR. For example, the establishment of a Safety Mode ($DP_{2.2.3}$) does not answer the needs defined by any of the system goals but ensures that certain safety properties are preserved during the operation of the robot. This is a case of a SC that results directly in the establishment of a level 2 function. $DP_{2.2.3}$ also serves to isolate the assumption MC6 that would otherwise affect $DP_{2.2.4}$ and $DP_{2.2.5}$.

## 4.2 Computer-Controlled Operations

This example analyzes part of the functionality of the Computer mode. Rather than again providing a walkthrough of the design process, the relevant semantic coupling issues are presented directly.

The high-level functional requirements corresponding to this mode are:

**MAPS-1.1.1.1:** MAPS shall generate appropriate commands to the Motor Controller to traverse the route provided by the Planner.

**MAPS-1.1.1.1.1:** MAPS shall inform the Planner of the success or failure of the route traversal and the reason for any failure.

**MAPS-1.1.1.1.2:** While in Computer Mode, all joystick deflections shall be ignored and the operator shall be informed when this occurs.

**MAPS-1.1.1.2:** While in Computer Mode, MAPS shall maintain information about the position of the robot. *Rationale: MAPS will use the position information to correct the robot trajectory and to determine when the final position has been reached.*

**MAPS-1.1.1.3:** In Computer Mode, MAPS shall determine when the stabilizer legs should be deployed (or retracted) and issue the appropriate commands.

### *4.2.1 Position Determination*

The first example analyzes the coupling created between the robot base motion control function and the position finding function. The level 2 SDPs that define the latter function are:

**MAPS-2.2.4.6: Scanner Query**. MAPS obtains position information from the laser scanner. Position is calculated by […] [↑MAPS-1.1.1.2, ↑LS2]

**MAPS-2.2.4.6.1:** Position determination occurs prior to the beginning of each route segment when operating in Computer Mode. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.2, ↑LS2, ↑LS2.1, ↑LS3] *Rationale: MAPS must know the robot position before it can send a new relative displacement command to the Motor Controller.*

**MAPS-2.2.4.6.2:** Position determination occurs following the completion of each route segment when operating in Computer Mode. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.2, ↑LS2, ↑LS2.1, ↑LS3] *Rationale: MAPS must know where the previous position command took the robot base. This information is used to detect errors and to provide feedback on previous actions in order to detect errors in carrying them out.*

The location system environmental assumptions used for the previous SDPs are:

**LS 2:** Upon request, the laser scanner will provide the current position of the robot in the form of global location coordinates with an accuracy of TBD.

**LS 2.1:** The laser scanner can only take position readings while the robot base is immobile.

**LS3:** The bar code targets remain within the line of sight of the scanner at all times.

The traceability matrix is derived from the FR-SDP links:

$$\begin{pmatrix} FR_{1.1.1.1} \\ FR_{1.1.1.2} \end{pmatrix} = \begin{bmatrix} 0 & * & * \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_{2.2.4.6} \\ SDP_{2.2.4.6.1} \\ SDP_{2.2.4.6.2} \end{pmatrix} \quad (8)$$

Equation (8) clearly represents the case of a coupled design. The origin of this coupling can be traced back to LS2.1. Because the scanner can only take position readings after the robot base has stopped, the motion and position finding functions are interdependent. Changes in some of the environmental assumptions can make the situation worse. For example, assume that LS3 does not hold anymore. This change would mean that in some cases, the scanner would not be able to find the bar code targets needed to perform its function. MAPS software engineers might be tempted to include additional motion controlling functionality to tackle this shortcoming. Unfortunately, the problem is nontrivial because it is hard to direct the robot to another position if there is a high degree of uncertainty about its current position. Despite having successfully isolated the LS environmental assumptions from the rest of the design (see Figure 4), the semantic coupling identified in equation (8) makes the entire system susceptible to changes in them. A possible solution to this problem is the substitution of the laser scanner by a Local Positioning System (LPS). This change would uncoupled the design and thus create a more robust system. It is unlikely that the decision to modify the robot would be based solely on this consideration; other factors (cost, equipment availability) are likely to play an important role as well. The key is to provide engineers with the information necessary to identify and evaluate those trade-offs of crucial importance to the long-term success of their system.

### *4.2.2 Robot Base Stabilization*

The final example involves the responsibilities of MAPS for robot base stabilization. The related part of the MAPS intent specification follows:

**MC1:** When commanded to do so, the motor controller will provide power to the motor, which will drive the robot wheels.

**GUI1:** The GUI will provide the operator with enough information about the status of the robot and the work area that the operator is able to avoid hazards. This information includes movements commanded by the Planner.

**PL1:** All automatic robot operations will be directed and coordinated by the Planner.

**SL1:** The stiff legs provide the robot base with the stability needed to perform all the currently anticipated robot arm operations. The stiff legs will be able to stabilize the robot during tile servicing (manipulator motion*).*

**SL2:** The stiff legs are not able to stabilize the robot while in motion..

**MA2:** The manipulator arm controller will provide information about the position of the arm directly to MAPS.

**SC6:** The mobile base must not move when the stabilizer legs are extended. *Rationale: Damage can occur to the robot if movement is attempted with the legs extended.*

**SC7:** The manipulator arm must not be extended when the stabilizers are retracted

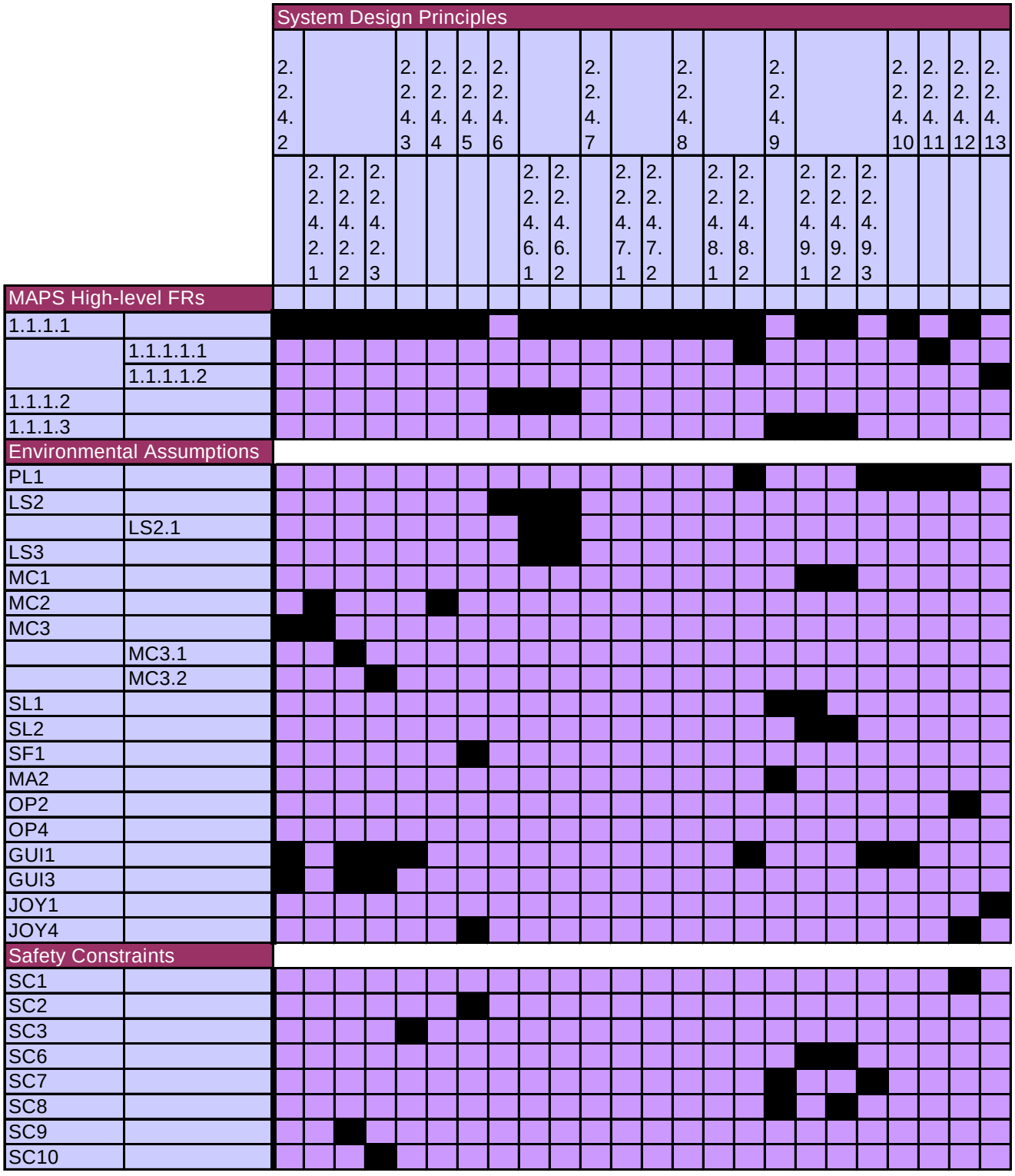
**SC8:** Stabilizer legs must not be retracted until the manipulator arm is fully stowed.

**MAPS-2.2.4.9: Leg Deployment and Retraction**. Upon reaching the final destination, MAPS deploys the stabilizer legs and later retracts them (upon the Planner's request) once the manipulator arm is stowed. [↑MAPS-1.1.1.3, ↑SL1, ↑MA2, ↑SC7, ↑SC8]

**MAPS-2.2.4.9.1:** After a move that has correctly positioned the robot base in the work area, MAPS turns off the wheel motors and then deploys the legs. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.3, ↑SL1, ↑SL2, ↑MC1, ↑SC6] *Rationale: The motor is turned off so no accidental motion command sets the base in motion with the stabilizers legs deployed.*

**MAPS-2.2.4.9.2:** When a move is commanded, MAPS retracts the stabilizer legs before turning on the wheel motors. [↑MAPS-1.1.1.1, ↑MAPS-1.1.1.3, ↑SL2, ↑MC1, ↑SC6, ↑SC8] *Rationale: see MAPS-2-2.4.9.1*

**MAPS-2.2.4.9.3:** In the event the stabilizer retraction or deployment fails, MAPS notifies the operator and the Planner of the failure. [↑GUI1, ↑PL1, ↑SC7] *Intent: The Planner is informed about the error so it does not send any arm motion commands. The operator is informed about the error so the failure can be diagnosed and repaired.*




| Legend | | |
|---|---|---|
| PL: Planner | SL: Stiff Legs | OP: Operator |
| LS: Location System | SF: Safety Circuit | GUI: Graphical User Interface |
| MC: Motor Controller | MA: Manipulator Arm | JOY: Joystick |

**Figure 4. Computer Control Mode Traceability Matrix.**

As before, the traceability matrix can be derived directly from the FR-SDP links:

$$\begin{pmatrix} FR_{1.1.1.1} \\ FR_{1.1.1.3} \end{pmatrix} = \begin{bmatrix} 0 & * & * \\ * & * & * \end{bmatrix} \begin{pmatrix} SDP_{2.2.4.9} \\ SDP_{2.2.4.9.1} \\ SDP_{2.2.4.9.2} \end{pmatrix} \quad (9)$$

As in the previous case, equation (9) indicates the design is coupled. The origin of this coupling is the need to satisfy safety constraints related to the instability of the robot base. Although on the surface the software logic appears to be relatively simple here, it can become quite complex due to the need to keep track of and control several unrelated subsystems and the related timing and fault tolerance issues. Changes to this logic could cause major disruptions to a project or rippling effects that delay deliverables or lead to budget overruns. Changes in this early system design phase could prevent these problems, and the semantic coupling information can assist the engineers in making the necessary tradeoff decisions. For example, a first step in decoupling this design might be to use a power interlock so the energy supply to the Motor Controller is shut off automatically whenever the manipulator arm power is turned on, thus reducing the safety responsibilities of the software.

## 5. CONCLUSIONS

The need for software engineering derives from the need to manage complexity. Complexity can be managed intellectually by partitioning, establishing hierarchies, and maximizing independence among components. Intent specifications (introduced previously) are a way of increasing intellectual manageability by using partitioning and hierarchical abstractions found by cognitive psychologists to be key to success in complex problem-solving activities. This paper has introduced and defined a concept of semantic coupling and shown how it can be operationalized using traceability matrices. Semantic coupling is related to the difficulty of changing system requirements and reducing coupling should reduce the rippling effects associated with requirements changes.

Semantic coupling at the higher system levels translates to structural coupling and cohesion at the software design and implementation levels and should also ease the structural decoupling process by decoupling functions or identifying coupled functionality at the higher levels. Any such decoupling involves design tradeoffs, but making these tradeoffs and their implications clear will be helpful to those generating system and software designs.

For small projects, the advantage of the approach lies in the intellectual processes that the domain experts must follow to produce the traceability matrices. For larger and more complex projects, the traceability matrices themselves play an important role. As the number of developers increases, creating a common convention to communicate coupling characteristics becomes increasingly important. During maintenance the need is even greater. Used in the Intent Specification framework, traceability matrices provide traditional functional traceability, but they also assist in system design and tradeoff evaluation, designing for safety and safety evaluation and assessment, support interface design, etc.

Providing techniques and tools to support the creation and navigation of large traceability matrices is a future research goal as is the more general goal of increasing our understanding of how to structure complex systems to allow us to stretch the limits of complexity of the systems we can build and maintain successfully.

## 6. REFERENCES


[1] Dowling, K.R., Bennett, R., Blackwell, M.,Graham, T., Gatrall, S., O'Toole, R., and Schempf, H. A Mobile Robot System for Ground Servicing Operations on the Space Shuttle. Cooperative intelligent robotics in space III, Proceedings of the Meeting, Boston, MA, Nov. 16-18, 1992 (A93-29101 10-54). Bellingham, WA, Society of Photo-Optical Instrumentation Engineers, 1992, p. 298-309.

[2] Glaser, R., Chi, M.T.H., and Farr, M.J. The Nature of Expertise. Erlbaum, Hillsdale, New Jersey, 1988.

[3] Jaffe, M.S., Completeness, Robustness, and Safety in Real-Time Software. Ph.D. Dissertation, University of California Irvine, 1988.

[4] Leveson, N. G. Intent Specifications: An Approach to Building Human-Centered Specifications. IEEE Transactions on Software Engineering, Vol. 26, No. 1, pp. 15-35, January 2000.

[5] Leveson N.G. Completeness in Formal Specification Language Design for Process-Control Systems. ACM Formal Methods in Software Practice, Portland, August 2000.

[6] Leveson, N.G., Reese, J.D. TCAS II Intent Specification. http://sunnyday.mit.edu/papers/intent.pdf

[7] Myers, G. J. Composite/Structured Design. Van Nostrand Reinhold Company, New York, NY, 1978.

[8] Rasmussen, J. The Role of hierarchical knowledge representation in decision making and system management. IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-15, no. 2, March/April 1985.

[9] Vicente, K.J., Christoffersen, K., and Pereklit, A..Supporting operator problem solving through ecological interface design. IEEE Transactions on Systems, Man, and Cybernetics, 25(4):529--545, 1995.

[10] Yourdon, E., and Constantine, L..Structured Design. Prentice Hall, Englewood Cliffs, N.J., 1979.