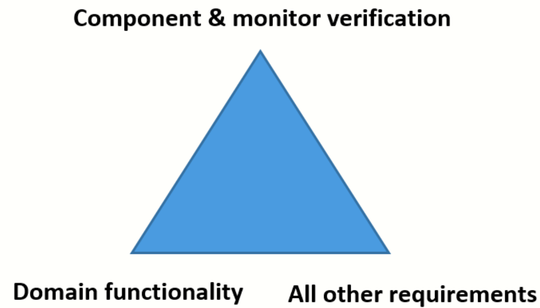
Triple Programming Appendices



Appendix A: Characterisitics of effective collaboration 2
Appendix B: Types and sources of requirements 3
Appendix C: Component Verification 4
Appendix D: Quality Goal Management 5
Appendix E: LiteRM XS quality model 6
Appendix F: Crosscutting requirements 11
Appendix G: Triple Programming introduction requirements 12

Send questions or comments to david@clearspecs.com

Appendix A: Characterisitics of effective collaboration

1. **Motivation**: Team members find a sense of individual purpose in the work itself or its results and believe the work matters to the company and its customers.

2. **Commitment**: Understand the advantages of collaboration. Resolve to collaborate despite the challenges of working with others.

3. **Flexibility**: Acknowledge diversity, for example in perception and decision making (Myers-Briggs). Understand and adjust to the challenges of diverse styles and viewpoints.

Effective collaboration entails safe co-creation

4. **Safety**: Psychological safety entails shared talk time, active listening and respect for concerns and alternative viewpoints, and high social sensitivity to the feelings and moods of others. It entails a team environment that encourages personal risk-taking in sharing thoughts and information. Concerns and negative behavior are identified and addressed. Conflict resolution tactics are identified and ranked.

5. **Defensiveness**: Identify defensive behaviors and examine the fears that trigger them.

6. **Trust**: When members trust one another, they work interdependently. They consider and explore one another's ideas. They recognize the value of constructive skeptics. Over time, members become familiar and even vulnerable.

7. **Candor**: Teams need to monitor progress and be honest about what's not working. Honesty enables strategy or performance adjustment and rework if necessary.

8. **Ownership**: Members understand they share responsibility for success and failure. There is no assignment of individual blame.

9. **Shared Goals**: Clearly specify the goals of a collaboration and co-create a strategy for achieving them.

10. **Structure**: Effective collaboration entails clearly-defined roles, responsibilities, goals, and strategies. Assignments should be flexible, but enable focused effort.

11. **Capability**: Each team member has or can quickly pick up the skills needed to carry out their responsibilities.

12. **Dependability**: Effective collaboration needs members to do their jobs and meet their deadlines. Team control replaces individual control.

"**A man's got to know his limitations**" - Dirty Harry
The devil's in the details
Software quality calls for inhuman accuracy

13. **Humility**: You may not know enough. Seek knowledge from others. To err is human. Seek skeptical analysis of your work. Team ego replaces individual ego.

14. **Support**: Members are expected to know their limitations and ask for help. Helping others is also expected. Identify accessible SMEs (subject matter experts) outside the team.

Appendix B: Types and sources of requirements

Types and subtypes of software requirements	Primary Sources
Quality goals	
Internal qualities e.g., readability	Quality specialists
External qualities e.g., reliability	Customers & Developers
Mixed qualities e.g., safety	Customers & Developers
Functions	
Domain functions	
interactive	
happy paths	Customers
unhappy paths	Developers
batch	Developers
autonomous	Developers
Quality support tactics	
internal quality supports e.g., self-checkers	Developers
external quality supports e.g., internationalization functions	Developers
mixed quality supports e.g., safeguards	Developers
misbehavior monitors	Developers
System functions e.g., backup	Developers
Data	Customers & Developers
Constraints	
Technical	
design e.g., no single point of failure, platforms, external interfaces and protocols	Developers
implementation e.g., coding standards, data restrictions	Quality specialists
verification e.g. test coverage	Verifiers
deployment e.g., mission configurations	Developers
Societal e.g., common practice	Quality specialists
Project e.g., deadlines	Project leads
Supplier attributes	Quality specialists

Appendix C: Component Verification

Component verification (CV) tries to answer the question: Will a plan, strategy, tactic, design, procedure, code, or data work as needed today and tomorrow? CV uses various forms of: **automated and manual analysis**, **skeptical review**, and **unit testing** to answer this question.

We assume that most developers have an acceptable understanding of domain functionality testing which is the best understood CV activity. Therefore, we will focus on other component-level SV targets and their verification. *We assume that internal, external, and mixed quality goals and the technical constraints have been validated.*

Other CV Targets

Other component-level targets of software verification	Primary Sources	Verification Activities
Quality achievement, verification, and monitoring strategies and tactics	Architects	Analysis and review
Functions		
Quality support tactics		
internal quality supports e.g., self-checkers	Developers	Review and test
external quality supports e.g., internationalization functions	Developers	Review and test
mixed quality supports e.g., safeguards	Developers	Review and test
misbehavior monitors	Developers	Review and test
Quality Support Data	Developers	Analysis and review
Technical Constraints		
design e.g., no single point of failure, platforms, external interfaces and protocols	Developers	Review and test
implementation e.g., coding standards, data restrictions	Quality specialists	Analysis and review
verification e.g. test coverage	Verifiers	Analysis and review

Component verifiers are also responsible for verifying system-level misbehavior monitors. Such monitors check whether the software continues to achieve its quality goals and reports changes in that achievement.

Appendix D: Quality Goal Management

Quality Goal Management (QGM) entails **specifying, achieving, and monitoring** an appropriate set of quality attribute goals. The full LiteRM quality model contains information about specifying, achieving, verifying and monitoring each quality goal. *We assume that using a tailored version of the full LiteRM quality model is the most efficient way to manage these goals*.

More information about the full LiteRM quality model and the model itself can be found at www.quality-aware.com/software-quality-KB.php including:

> "Simplistic Models Considered Harmful"
> Chapter 3: Quality Goals from **Understanding Requirements**

QGM is also responsible for the development of system-level misbehavior monitors. For example, a system-level safety monitor might check for unanticipated acceleration and trigger suitable action if such acceleration is detected.

Appendix E: LiteRM XS quality model

LiteRM XS quality model

INTERNAL quality attributes (32) -- directly visible **only** to developers

- *Basic quality group*
  - Understandability
    - Conceptual Integrity
      - Necessity
      - Consistency
      - Domain alignment
      - MODULARITY subgroup
        - Cohesion
        - Coupling
    - Code Readability
      - Essential complexity
      - Organization
      - Predictability
  - VERIFIABILITY subgroup
    - Testability (externals and mixed)
      - Observability
        - Monitorability
      - Controllability
      - REPAIRABILITY subgroup
        - Diagnosability
        - Modifiability
        - Restorability
      - Recoverability
    - Reviewability (all)
    - Analyzability (all)
    - Measurability (all)
  - COMPLIANCE subgroup
    - Laws and regulations
    - Design and coding standards
    - Verification guidelines
- *Internal Behavior group*
  - Internal PERFORMANCE subgroup
    - Capacity
    - Efficiency
- *Internal Durability group*
  - Interversion compatibility
  - Configuration reusability
    - Component reusability
  - ADAPTABILITY subgroup
    - Portability
    - Enhanceability

EXTERNAL quality attributes (22) -- fully visible to users

- *Facilitation quality group*
  - Ease of configuring, updating, and operating
  - Ease of installing and uninstalling
  - Ease of learning
    - Ease of use
      - EASE OF ACCESS subgroup
        - Ease of access in different situations
        - Ease of access with different disabilities
      - Error resistance
      - Functional Integrity
      - COMPATIBILITY subgroup
        - Platform compatibility
        - Application compatibility
          - Interoperability
  - Ease of managing
    - Ease of auditing
      - Ease of detecting dishonesty

- *External Behavior group*
  - Availability
    - Reliability
      - Correctness
        - Accuracy
        - Precision
  - External PERFORMANCE subgroup
    - Responsiveness
    - Throughput

- *External Durability group*
  - Internationalizability

MIXED quality attributes (14) -- partially visible to users, usually upon failure

- *Mixed Behavior group*
  - Software trustworthiness
    - Safety
      - Dependability
        - Data trustworthiness
          - Sources trustworthiness
          - Processes trustworthiness
        - Resiliency
          - Robustness
            - Fault tolerance
            - Survivability
          - Security
            - Privacy
            - Resource security
      - [External Behavior group]

- *Mixed Durability group*
  - Scalability

Each quality attribute is associated with a fixed set of over 30 properties. In the full LiteRM quality model, some of these properties have values. More values can be added based on successful specification, achievement, verification, or monitoring.

Safety [An example of properties and values]

Definition The ability of a system to do little or no harm to valuable assets

Software subfield safety engineering

Assumptions/Rationale

1. Safety is a fragile quality because it depends on many other qualities and the accurate identification of safety hazards.
2. Ultra-safety requirements (e.g. MISRA SIL 4) should be defined by a set of required achievement constraints.

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Ratio of hazards added during HA technical review to hazard count after HA technical review
- Ratio of safeguard defects identified during a technical review to number of safeguards
- Ratio of safeguard defects found during testing to number of safeguards

Operational Measures

- Time since last "dangerous" failure or defect
- Number of "dangerous" failures or defects detected per time interval
- Greatest harm from a harmful event
- Shortest harmful event free duration
- Longest harmful event free duration
- Expected length of harmful event free duration
- Expected rate of harmful events
- Ratio of actual loss to acceptable loss in a duration
- Estimated residual risk

Aspect of software trustworthiness

Supported by – always or sometimes dependability, resiliency, **ease of learning**

> **Note:** While safety-critical functionality may be supported by these qualities, at the same time they may conflict with non-safety-critical functionality. For example, availability supports dependability, but it may cause non-safety-critical functionality to be sacrificed so the system can continue to operate in a safe, but degraded mode.

Conflicts with efficiency, interoperability, adaptability qualities

Threats [identify using hazard analysis - this is an aggregate quality that includes one quality for each specific hazard e.g., safe *from death by electrocution* (the hazard)]

Mitigations [identify after identifying hazards]

Other achievement tactics

- identify valuable assets and hazards
- identify safety-critical and safety-related functions and constraints needed for safety
  e.g. "The Fire Detection System shall detect smoke above X ppm within 5 seconds."
- isolate and protect safety-critical functions
- guard safety-critical functions with explicit conditions i.e. never with defaults such as "otherwise"
- identify safety-critical users
- eliminate or mitigate hazards i.e. identify appropriate control actions
- effectively execute control actions and receive accurate and sufficient feedback
- alert users to dangerous actions with rotating warning messages
- precede each dangerous action with a delay so user can change their mind and cancel
- limit complexity
- design interfaces that prevent and detect user errors
- use warning labels and messages when appropriate
- use specified tactics for higher-level SILs

Verification tactics

- review hazards and mitigations for completeness and effectiveness during a safety audit
- thoroughly test each safeguard
- measure and track time since last "dangerous" failure or defect and number of "dangerous" failures or defects,
- verify all supporting qualities
- use self-checks
- monitor for misbehavior e.g., unanticipated acceleration
- monitor system state to make sure safety-critical and safety-related functions are active

Elicitation Questions

- What valuable assets are at risk?
- Which functions are safety-critical or safety-related?
- Who/What can perform these safety-critical or safety-related functions and under what conditions?
- What harm can the system or its actors possibly do?
- What can mitigate these hazards?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

"Quality Attributes" Technical Report CMU/SEI-95-TR-021 Chapter 6

Engineering a Safer World

Software Safety Primer

MISRA Report 2 on Integrity

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Kind = homogeneous composite attribute

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Retired>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Appendix F: Crosscutting requirements

Every developer (and manager) should understand that code successfully performing a domain function is just the beginning of quality software [Fig. 1]. Quality goals need quality support code, such as input verification, exception handling, logging, safeguards, security guards, encryption, and software test points. This quality support code impacts i.e., crosscuts, many components.

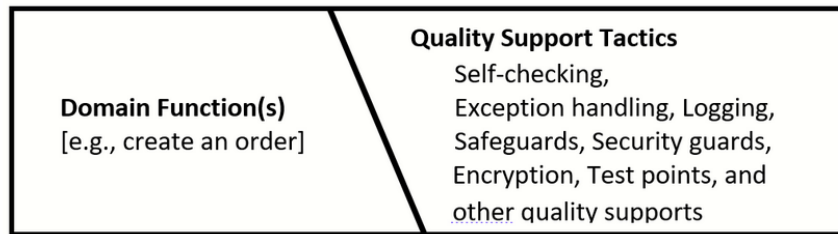


Fig. 1. Domain functionality is just the beginning

We distinguish three types of crosscutting requirements: **ubiquitous**, **homogenous**, and **heterogeneous**.

Some requirements, such as understandability, are associated with ubiquitous crosscuts. These requirements restrict designs and most lines of code. A ubiquitous crosscut restricts all other crosscuts, including other ubiquitous crosscuts e.g., understandability supports testability.

The implementation of a homogenous crosscut does the same thing everywhere it appears. For example, a homogenous logging routine may appear at the exit of every module, but always collects and stores the same type of information.

The implementation of a heterogeneous crosscut does different things everywhere it appears because its behavior is context-sensitive. The variety of security threats means that different security guards will be needed for each threat i.e., security entails heterogeneous crosscuts.

Quality goals should be identified early along with their achievement and verification strategies. If domain functions are coded before many quality goals are identified, there will be significant **quality debt** associated with these component because of missing quality support code.

Appendix G: Triple Programming introduction requirements

Introducing triple programming has **training, guidance, and measurement requirements**.

[Collaboration]

1. Developer pre-surveys
2. Overview sheet (**exists – Appendix A**)
3. Overview (presentation → video)
4. Project experience, with guidance and periodic assessments
5. Developer post-surveys

Apple CEO Tim Cook on Collaboration https://www.youtube.com/watch?v=EZPYLZ7I6gs

Secrets Of Successful Teamwork: Insights From Google https://www.youtube.com/watch?v=hHIikHJV9fI

Douglas Stone & Sheila Heen **Thanks for the Feedback** Penguin 2014

[Component & Monitor Verification]

1. Overview sheet (**exists – Appendix C**)
2. Overview (presentation → video)
3. Project experience, with guidance and periodic assessments
4. Developer post-surveys

[Quality Goal Management]

1. Overview sheet (**exists – Appendix D**)
2. Essentially-complex quality attribute model (**exists**)
3. Overview (presentation → video) **Videos 1, 4, & 5 exist**
4. Project experience, with guidance and periodic assessments
5. Developer post-surveys

"Simplistic Models Considered Harmful" (**exists**)
Chapter 3: Quality Goals from **Understanding Requirements** (**exists**)
"LiteRM Quality Model" freely available at www.quality-aware.com/software-quality-KB.php (**exists**)

Triple Programming

1. Overview sheet (**exists – tripleprogramming.com home page**)
2. Overview (presentation → video)
3. Project experience, with guidance and periodic assessments
4. Developer post-surveys
5. Leadership post-surveys