Accepted in principle at meeting 53. Check that the corresponding descriptions support the guidance.

These guidance items were identified by Erhard as being general enough for TR 1. After review, the following items were managed as documented below.

6.6

* Always respect the implied unit systems, when converting explicitly from one numeric type to another.

(This was added to Part 1, can be removed from Part 2)

6.9

* Do not suppress the checks provided by the language.

(This was added to Part 1, can be removed from Part 2)

6.35

* Alternatively, monitor the depth of the recursion such as by passing a recursion depth value that is incremented for each level of recursion, and use a subtype constraint or explicit comparison against a maximum depth limit to trigger handling of the situation.

(Added, can be removed from Part 2)

6.44

* Preceed downcasts by a class-wide membership test as needed to avoid possible exceptions.
* Use type invariants where allowed to detect semantic violations caused by upcasts.

(This was added, can be removed from Part 2)

6.52

* Do not suppress language defined checks.

(This was already there, so can be removed from Part 2)

* If language-defined checks must be suppressed, use static analysis to prove that the code is correct for all combinations of inputs.

(This was already there, so can be removed from Part 2)

* If language-defined checks must be suppressed, use explicit checks at appropriate places in the code to ensure that errors are detected before any processing that relies on the correct values.

(Added to Part 1, can be removed from Part 2)

6.55

* For situation where order of evaluation or number of evaluations is unspecified, use only operations with no side-effects, or idempotent behaviour, to avoid the vulnerability.

(This was added to Part 1, can be removed from Part 2)

6.57

* Minimize use of predefined numeric types whose ranges and precisions are implementation defined. Instead, use types whose ranges and precision are guaranteed.

(Moved to 6.2 of Part 1, irrelevant change for Part 2)

6.60

* Where possible do not use forced termination

(This was added to Part 1, can be removed from Part 2)

6.63

* On a single processor, make use of a scheduling regime based on ceiling protocols, this is guaranteed to be deadlock free (if the tasks and resources are assigned the correct priorities – a static property that can be checked offline).
* For multicore, consider assigning all interacting tasks to the same CPU then treat each such group as a separate independent entity.
* Minimize the use of dynamic priorities and dynamic ceiling priorities (so that the static values can be verified)

(This was added to Part 1, can be removed from Part 2)