Chapter 12

Impact Analysis

Overview

- Impact Analysis (Reason, Traceability, Ripple Effect)
- Impact analysis process
- Dependency based impact analysis
- Ripple Effect
- Change Propagation Model

Impact Analysis

Impact analysis is the process of identifying the components that are impacted by the change request.

Impact analysis enables understanding and implementing changes in the system. Potential effects of the proposed changes are made visible by performing impact analysis.

In addition, it is used in estimating cost and planning a schedule.

Reason for Impact Analysis

- To estimate the cost of executing the change request. Before we fix or add anything new, we want to know how much it will cost so we're not surprised later.

- To determine whether some critical portions of the system are going to be impacted due to the requested change.

- To understand how items of change are related to the structure of the software. We want to see how the change fits into the big picture, and if it's going to affect other parts of the system.

- To determine the portions of the software that need to be subjected to regression testing after a change is effected – To ensure nothing else was broken while making a change.

Impact Analysis Traceability

Def: Traceability is the ability to trace between software artifacts generated and modified during the software product life cycle. Thus, traceability helps software developers understand the relationships among all the software artifacts in a project.

Examples of such entities are design and source code.

There are two broad kinds of traceability:
(i) horizontal (external) traceability; and
(ii) vertical (internal) traceability.

Traceability of artifacts between different models is known as external traceability, whereas internal traceability refers to tracing dependent artifacts within the same model. Internal traceability primarily focuses on source code artifacts.

Impact Analysis and Ripple effect

Ripple effect means that a modification to a single variable may require several parts of the software system to be modified. The concept of ripple effect has relevance in software evolution because it concerns changes and their effects

Measurement of ripple effects can be provided by the following information about an evolving software system:

(i) between successive versions of the same system, measurement of ripple effect will tell us how the software's complexity has changed;

(ii) when a new module is added to the system, measurement of ripple effect on the system will tell us how the software's complexity has changed because of the addition of the new module.

Impact Analysis Process

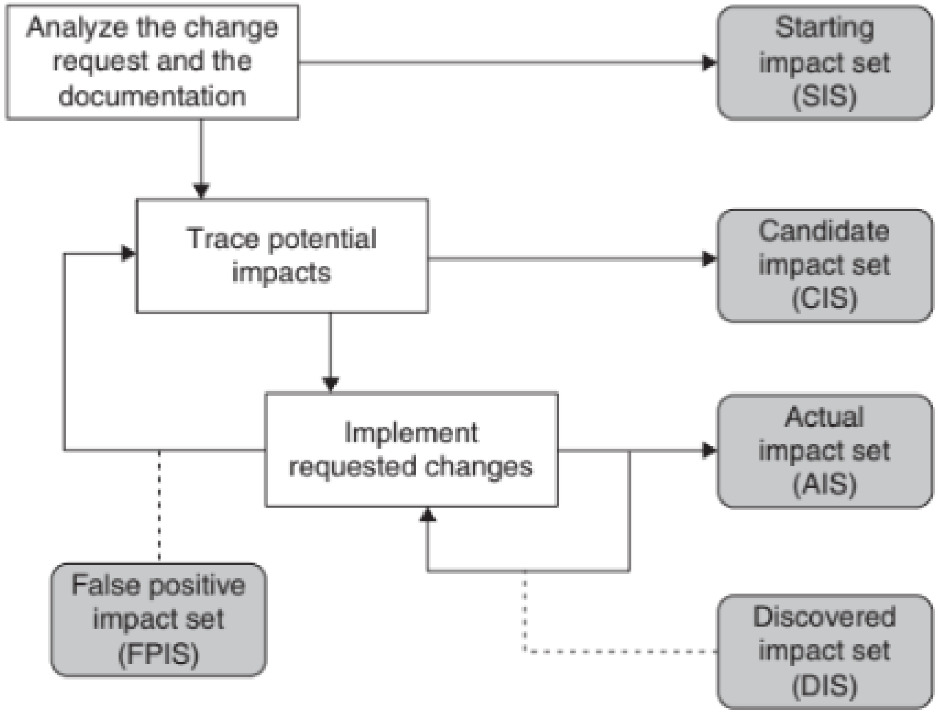

FIGURE 6.1 Impact analysis process. From Reference 6. © 2008 IEEE

FPIS = parts of the software that are mistakenly identified as being affected by a change — even though they actually aren't.

Impact Analysis Process

Figure 6.1 depicts a process of impact analysis.

1. The process begins by analyzing the CR, the source code, and the associated documentation to identify an initial set, called starting impact set (SIS), of software objects that are likely to be affected by the required change.

2. To discover additional elements to be affected by the CR, the SIS is analyzed. The union of SIS and the new set generated by analyzing SIS is the candidate impact set (CIS)

3. An actual impact set (AIS) is obtained after the change is actually implemented. Given that one can implement a CR in many ways, the AIS set is not unique.
Ideally, AIS should be equal to SIS ∪ DIS \ FPIS, where ∪ denotes set union and \ denotes set difference.

4. A discovered impact set (DIS) represents the collection of all those newly discovered elements, and it indicates an underestimation of impacts of the change.

Impact Analysis Process

5. Some members of CIS may not be actually impacted by the CR, and the group of those entities is known as false positive impact set (FPIS). FPIS indicates an overestimation of impacts.

The **error** in impact estimation can be computed as **(|DIS| + |FPIS|) / |CIS|**.

Impact Analysis Process

1. Identifying the SIS:

   Impact analysis begins with identifying the SIS. The CR specification, documentation, and source code are analyzed to find the SIS.

   It takes more effort to map a new CR's "concepts" onto source code components (or objects).

   There are several methods to identify concepts, or features, in source code. The "grep" pattern matching utility available on most Unix systems and similar search tools are commonly used by programmers

   The technique often fails when the concepts are hidden in the source code, or when the programmer fails to guess the program identifiers.

Impact Analysis Process

1. Identifying the SIS:

   The software reconnaissance methodology proposed by Wilde and Scully is based on the idea that some programming concepts are selectable, because their execution depends on a specific input sequence. Selectable program concepts are known as features.

   By executing a program twice, one can often find the source code implementing the features:

   (i) execute the program once with a feature and once without the feature;
   (ii) mark portions of the source code that were executed the first time but not the second time;
   (iii) the marked code are likely to be in or close to the code implementing the feature.

Impact Analysis Process

1. Identifying the SIS:

   Chen and Rajlich proposed a dependency-graph-based feature location method for C programs.

   The component dependency graph is searched, generally beginning at the main().

   Functions are chosen one at a time for a visit. The maintenance personnel reads the documentation, code, and dependency graph to comprehend the component before deciding if the component is related to the feature under consideration.

   The C functions are successively explored to find and understand all the components related to the given feature.

Impact Analysis Process

2. Analysis of Traceability Graph:

   Whenever change is proposed, it is necessary to analyze the traceability graphs in terms of its complexity and size to assess the maintainability of the system. By means of an example, we explain the traceability.

   By means of an example, we explain the traceability links and graphical relationships among related work products (see Figure 6.2). The graph is so constructed that reveals the relationships among work products. Specifically, the graph shows the horizontal traceability of the system.

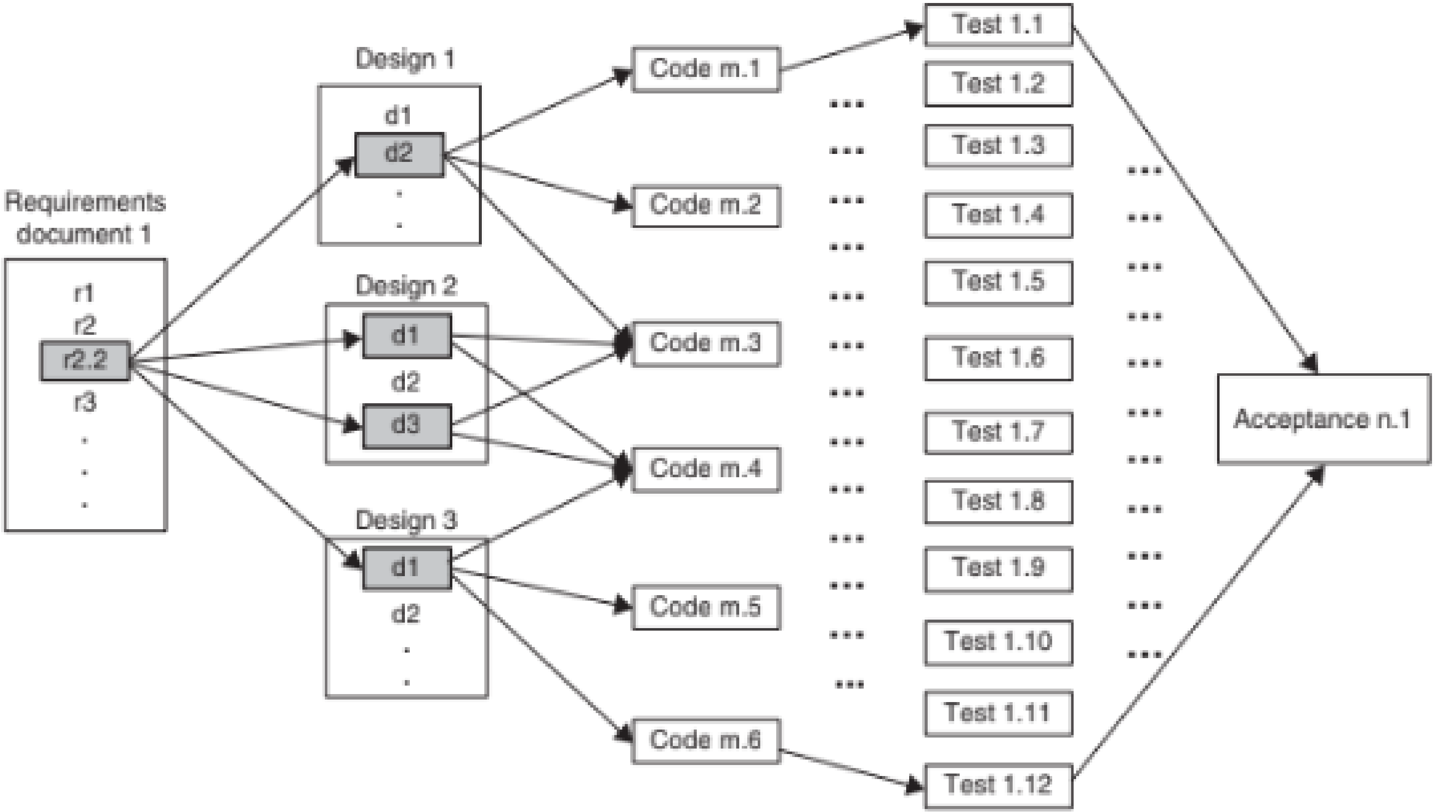

FIGURE 6.2 Traceability in software work products. From Reference 22. © 1991 IEEE

Impact Analysis Process

2. Analysis of Traceability Graph:

   In Figure 6.3, each category of nodes is represented by a silo, and additional edges can be found within a silo. The edges within a silo represent vertical traceability for the kind of work product represented by the silo. Vertical traceability has been represented by solid lines, whereas horizontal traceability by dashed lines.

   As work products change, both the vertical traceability and horizontal traceability are likely to change. The change to vertical traceability is assessed by considering the complexity and size of the vertical traceability graph within each silo. A common measure of complexity of a graph is the well-known Cyclomatic complexity. It may be noted that vertical traceability metrics are product metrics and those metrics reflect the effect of change on each product.

   On the other hand, process metrics are useful in examining horizontal traceability. To understand changes in horizontal traceability, it is necessary to understand:
   (i) the relationships among the work products; and (ii) how work products relate to the process as a whole.

Impact Analysis Process

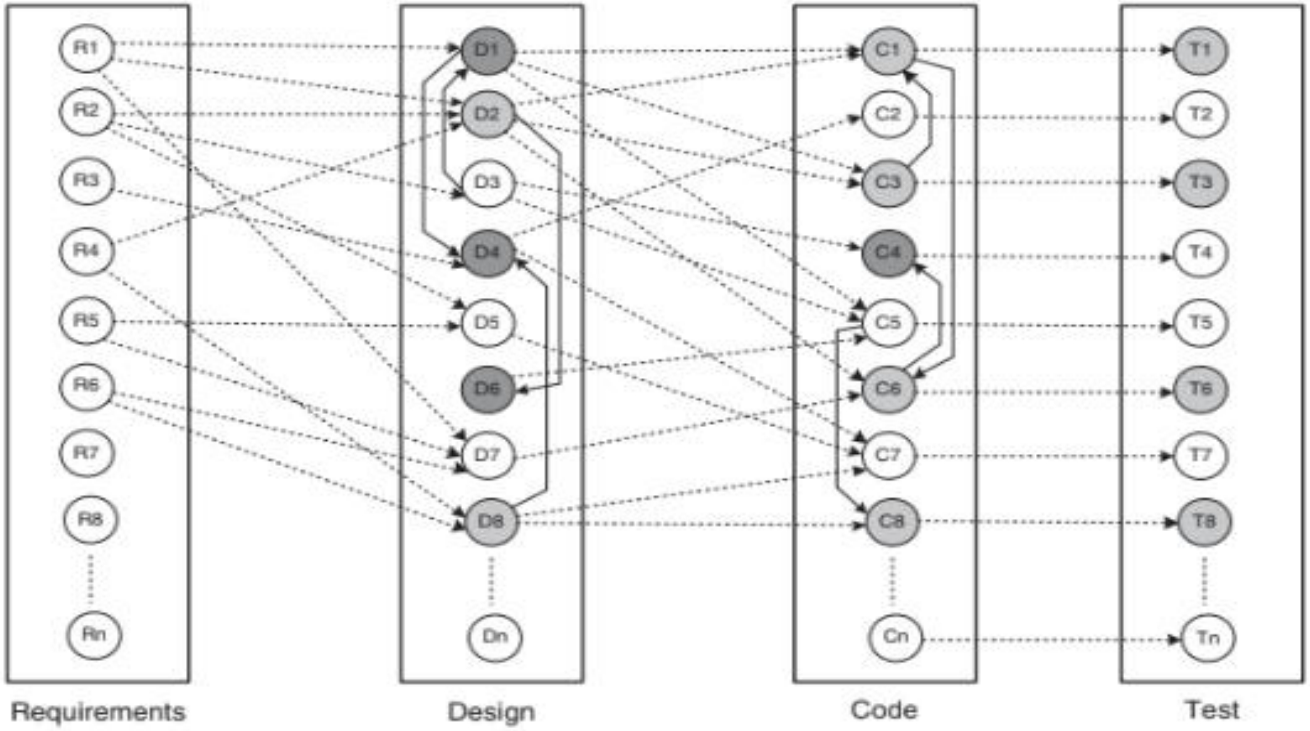

FIGURE 6.3 Underlying graph for maintenance. From Reference 22. © 1991 IEEE

Impact Analysis Process

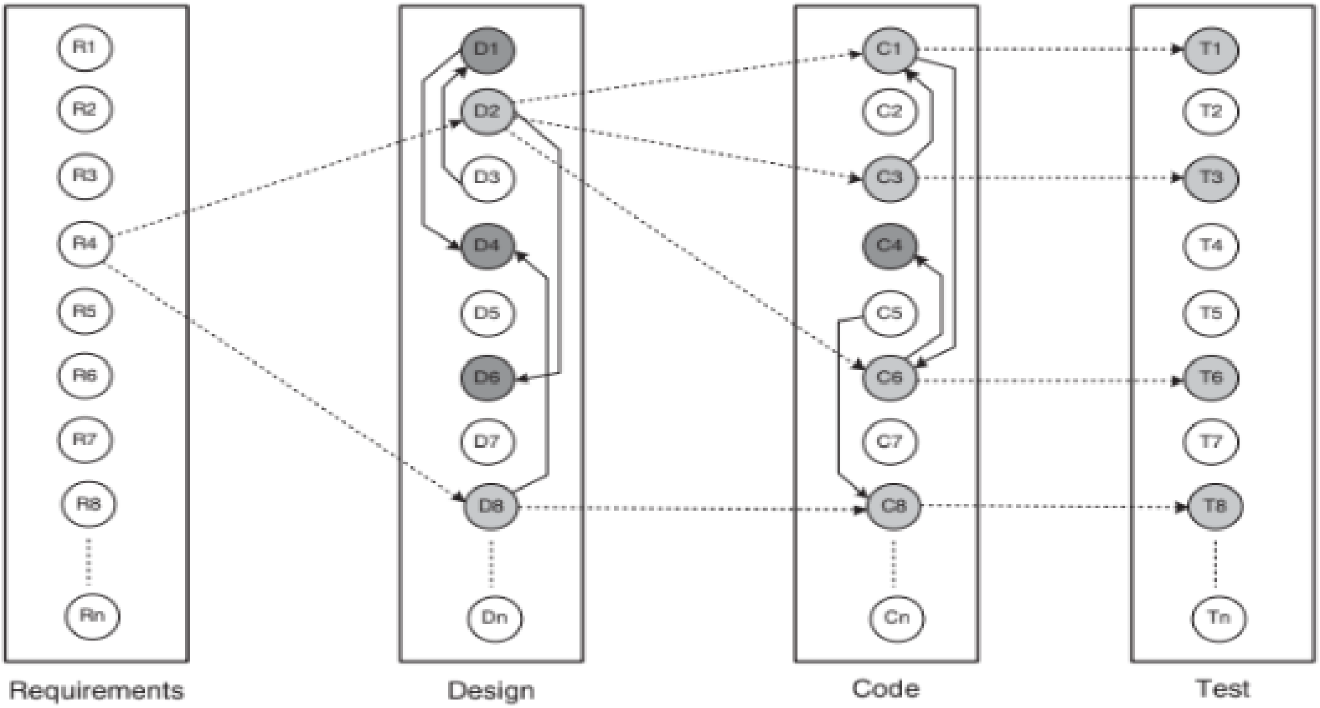

FIGURE 6.4 Determine work product impact. From Reference 22. © 1991 IEEE

Impact Analysis Process

3. Identifying the Candidate Impact Set

A CIS is identified in the next step of the impact analysis process. The SIS is augmented with software lifecycle objects (SLOs) that are likely to change because of changes in the elements of the SIS.

Changes in one part of the software system may have direct impacts or indirect impacts on other parts

Direct impact: A direct impact relation exists between two entities, if the two entities are related by a fan-in and/or fan-out relation.

Indirect impact: If an entity A directly impacts another entity B and B directly impacts a third entity C, then we can say that A indirectly impacts C. Relation would look like this: A → B → C

Impact Analysis Process

3. Identifying the Candidate Impact Set

Each SLO represents a software artifact connected to other artifacts. The artifacts can be arbitrary entities, ranging from a requirement of the entire system to the definition of a variable.

Dependencies among SLOs are represented by arrows. In the figure, SLO1 has an indirect impact from SLO8 and a direct impact from SLO9.

The in-degree of a node i reflects the number of known nodes that depend on i.

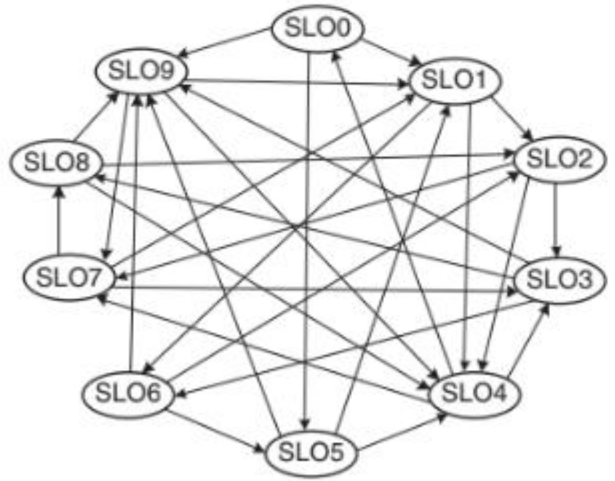

FIGURE 6.5 Simple directed graph of SLOs. From Reference 12. © 2002 IEEE

Impact Analysis Process

3. Identifying the Candidate Impact Set

The connectivity matrix is constructed by considering the SLOs and the relationships shown in Figure 6.5.

TABLE 6.1 Relationships Represented by a Connectivity Matrix

	SLO0	SLO1	SLO2	SLO3	SLO4	SLO5	SLO6	SLO7	SLO8	SLO9
SLO0		x				x				x
SLO1			x		x		x			
SLO2				x	x			x		
SLO3							x		x	x
SLO4	x			x				x		
SLO5		x			x					x
SLO6			x			x				x
SLO7		x		x					x	
SLO8			x		x					x
SLO9		x			x			x		

Source: From Reference 12. © 2002 IEEE.

Impact Analysis Process

3. Identifying the Candidate Impact Set

A reachability graph can be easily obtained from a connectivity matrix.

A reachability graph shows the entities that can be reached

TABLE 6.2 Relationships Represented by a Reachability Matrix

	SLO0	SLO1	SLO2	SLO3	SLO4	SLO5	SLO6	SLO7	SLO8	SLO9
SLO0		x	x	x	x	x	x	x	x	x
SLO1	x		x	x	x	x	x	x	x	x
SLO2	x	x		x	x	x	x	x	x	x
SLO3	x	x	x		x	x	x	x	x	x
SLO4	x	x	x	x		x	x	x	x	x
SLO5	x	x	x	x	x		x	x	x	x
SLO6	x	x	x	x	x	x		x	x	x
SLO7	x	x	x	x	x	x	x		x	x
SLO8	x	x	x	x	x	x	x	x		x
SLO9	x	x	x	x	x	x	x	x	x	

Source: From Reference 12. © 2002 IEEE.

Impact Analysis Process

3. Identifying the Candidate Impact Set

The dense reachability matrix of Table 6.2 has the risk of over-estimating the CIS. To minimize the occurrences of false positives, one might consider the following two approaches.

Distance-based approach: In this approach, SLOs which are farther than a threshold distance from SLO i are considered not to be impacted by changes in SLOW i. In Table 6.3, the concept of distance has been introduced in the analysis. One can estimate the scope of the ripple by augmenting Warshall's algorithm with data about the nodes traversed so far.

Incremental approach: In this approach, the CIS is incrementally constructed. For every SLO in the SIS, one considers all the SLOs interacting with it, and only SLOs that are actually impacted by the change request are put in the CIS. The identification process is recursively executed until all the impacted SLOs are identified.

Impact Analysis Process

Several metrics are defined in the literature to evaluate the impact analysis process. Here, we discuss two traditional information retrieval metrics: recall and precision.

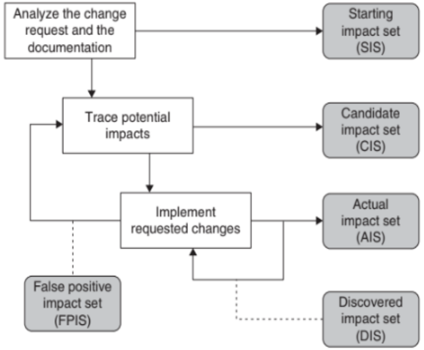

FIGURE 6.1 Impact analysis process. From Reference 6. © 2008 IEEE

Recall: It represents the fraction of actual impacts contained in CIS, and it is computed as the ratio of **|CIS ∩ AIS| to |AIS|**. The value of recall is 1 when DIS is empty.

Meaning, from the total actual set, how many of them was chosen corretly as a candidate

Precision: It represents the fraction of candidate impacts that are actually impacted, and it is computed as the ratio of **|CIS ∩ AIS| to |CIS|**. For an empty FPIS set, the value of precision is 1.

Meaning, from the candidate set, how many of them actually impacted

Impact Analysis Process

Adequacy and Effectiveness are two key aspects of any impact analysis approach

Adequacy: Adequacy of an impact analysis approach is the ability of the approach to identify all the affected elements to be modified. Ideally, **AIS ⊆ CIS**. Adequacy is repressed in terms of a performance metric called inclusiveness, as follows.

$$\mathit{Inclusiveness} = \begin{cases} 1 & \text{if AIS } \subseteq \text{ CIS} \\ 0 & \text{otherwise} \end{cases} .$$

The concept of adequacy is essential to assessing the quality of an impact analysis approach.

A method is considered adequate if it doesn't miss anything important that needs to be updated or fixed.

Impact Analysis Process

Effectiveness: The ability of an impact analysis technique to generate results, that actually benefit the maintenance tasks, is known as its effectiveness.

Effectiveness is expressed in terms of three fine-grained characteristics as follows.
– Ripple-sensitivity
– Sharpness
– Adherence

Impact Analysis Process

Ripple-sensitivity implies producing results that are influenced by ripple effect.

The set of objects that are directly affected by the change is denoted by DISO (directly impacted set of objects), and it is also known as primary impacted set (PIS). Similarly, the set of objects that are indirectly impacted by the change is denoted by IISO (indirectly impacted set of objects), and it is also known as the secondary impacted set (SIS).

The cardinality of IISO is an indicator of ripple effect.

The software maintenance personnel expect that the cardinality of IISO is not far from the cardinality of DISO. Therefore, the concept of Amplification, as defined below, is used as a measure of Ripple-sensitivity.

$$Amplification = \frac{|IISO|}{|DISO|} \longrightarrow 1,$$

where | . | denotes the cardinality operator.

Impact Analysis Process

Sharpness is the ability of an impact analysis approach to avoid having to include objects in the CIS that need not be changed. Sharpness is expressed by means of Change Rate as defined below.

$$ChangeRate = \frac{|\, CIS \,|}{|\, System \,|}.$$

It may be noted that CIS is included in "System", and Change Rate falls in the range from 0 to 1. For Sharpness to be high, we must have Change Rate ≪ 1.

Impact Analysis Process

Adherence is the ability of the approach to produce a CIS which is as close to AIS as possible. A small difference between CIS and AIS means that a small number of candidate objects fail to be included in the actual modification set. Adherence is expressed by S-Ratio as follows:

$$S\text{-}Ratio = \frac{|AIS|}{|CIS|}.$$

If the impact analysis approach is adequate, AIS is included in CIS, and S-Ratio takes on values in the range from 0 to 1. Ideally, the S-Ratio is equal to 1.

Dependency based Impact Analysis

https://www.apriorit.com/qa-blog/252-impact-analysis

Dependency-based Impact Analysis

Dependency-based impact analysis techniques identify the impact of changes by analyzing syntactic dependencies, because syntactic dependencies are likely to cause semantic dependencies.

Two traditional impact analysis techniques are explained in this section. The first technique is based on call graph, whereas the second one is based on dependency graph

Call Graph:

A call graph is a directed graph in which a node represents a function, a component, or a method, and an edge between two nodes A and B means that A may invoke B.

Dependency-based Impact Analysis

Call Graph:
Let P be a program, G be the call graph obtained from P, and p be some procedure in P.

A key assumption in the call-graph-based technique is that some change in p has the potential to impact changes in all nodes reachable from p in G.

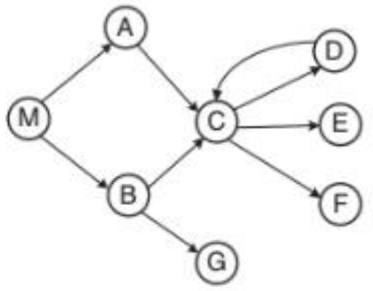

FIGURE 6.7 Example of a call graph. From Reference 26. © 2003 IEEE

Dependency-based Impact Analysis

Call Graph:

Call-graph-based approach to impact analysis suffers from many disadvantages as follows:

- A call graph represents the potential calls by a single procedure, while ignoring the dynamic aspects. Consequently, impact analysis based on call graphs can produce an imprecise impact set. For example, in Figure 6.7, one cannot determine the conditions that cause impacts of changes to propagate from M to other procedures.

- Generally, a call graph captures no information flowing via returns. Therefore, impact propagations due to procedure returns are not captured in the call-graph- based technique. Suppose that in Figure 6.7, D is modified and control returns to C. Now, following the return to C, it cannot be inferred whether impacts of changing E propagates into none, both, A, or B.

Dependency-based Impact Analysis

Call Graph:

To address the aforementioned issues, Law and Rothermel defined a technique called path-based dynamic impact analysis that uses whole path profiling to estimate the effects of changes.

In this approach, if a procedure p is changed, then one considers the impact that is likely to propagate along those executable paths that are seen to be passing through p. As a result, any procedure, that is invoked after p but still appears on the call stack after p terminates, is assumed to be potentially impacted.

Dependency-based Impact Analysis

Call Graph:

M B r A C D r E r r r r x.

FIGURE 6.8 Execution trace

Let us consider an execution trace as shown in Figure 6.8. The trace corresponds to a program whose call graph is shown in Figure 6.7. In the figure, r and x represent function returns and program exits, respectively.

Let procedure E be modified. The impact of the modification with respect to the given trace is computed by forward searching in the trace to find:

(i) procedures that are indirectly or directly invoked by E; and

(ii) procedures that are invoked after E terminates. One can identify the procedures into which E returns by performing backward search in the given trace. For example, in the given trace, E does not invoke other entities, but it returns into M, A, and C. Due to a modification in E, the set of potentially impacted procedures is {M, A,C, E}.

Dependency-based Impact Analysis

Program Dependency Graph:

In the program dependency graph (PDG) of a program:
(i) each simple statement is represented by a node, also called a vertex; and
(ii) each predicate expression is represented by a node.

There are two types of edges in a PDG: data dependency edges and control dependency edges.

In the following figure Data dependencies are shown as solid edges, whereas control dependencies are shown as dashed edges.

Dependency-based Impact Analysis

Program Dependency Graph:

```
      begin
S1:       read(X)
S2:       if (X < 0)
          then
S3:           Y = f1 (X);
S4:           Z = g1 (X);
          else
S5:           if (X = 0)
              then
S6:               Y = f2 (X);
S7:               Z = g2 (X);
              else
S8:               Y = f3 (X);
S9:               Z = g3 (X);
              end_if;
          end_if;
S10:      write(Y);
S11:      write(Z);
     end.
```

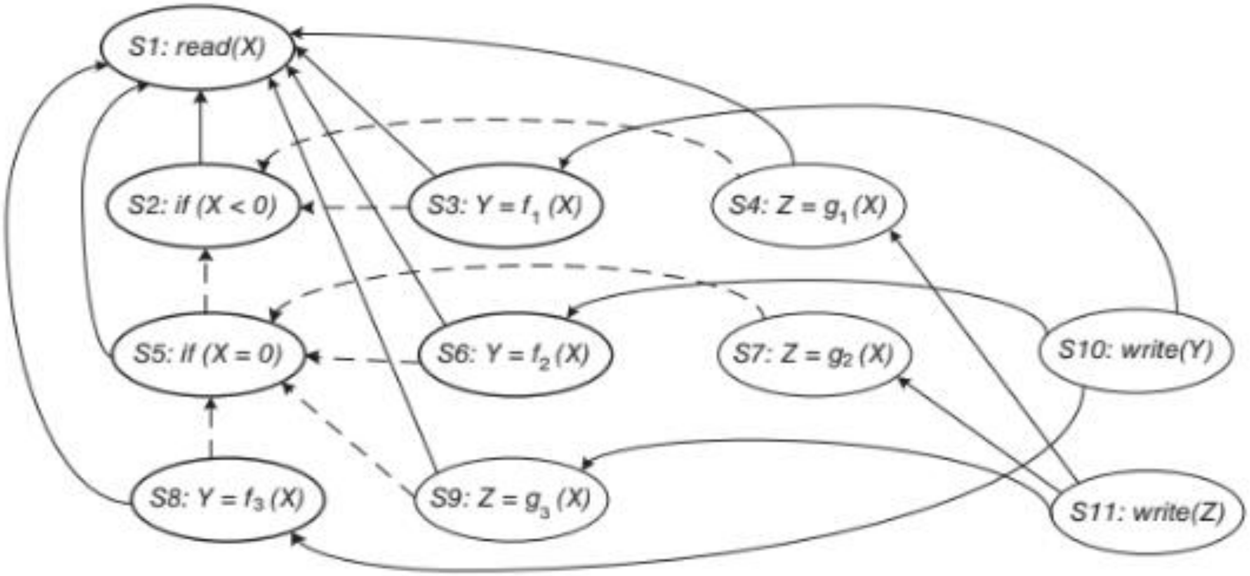

FIGURE 6.10 Program dependency graph of the program in Figure 6.9

Dependency-based Impact Analysis

Static Program Slice:

A static program slice is identified from a PDG as follows:
(i) for a variable var at node n, identify all reaching definitions of var; and
(ii) find all nodes in the PDG which are reachable from those nodes.

The visited nodes in the traversal process constitute the desired slice.

Consider the program in Figure 6.9 and variable Y at S10. First, find all the reaching definitions of Y at node S10—and the answer is the set of nodes {S3, S6, and S8}.
Next, find the set of all nodes which are reachable from {S3, S6, and S8}—and the answer is the set {S1, S2, S3, S5, S6, S8}.

Dependency-based Impact Analysis

Program Dependency Graph:

begin
S1: *read*(X)
S2: *if* ($X < 0$)
then
S3: $Y = f_1(X)$;
S4: $Z = g_1(X)$;
else
S5: *if* ($X = 0$)
then
S6: $Y = f_2(X)$;
S7: $Z = g_2(X)$;
else
S8: $Y = f_3(X)$;
S9: $Z = g_3(X)$;
end_if;
end_if;
S10: *write*(Y);
S11: *write*(Z);
end.

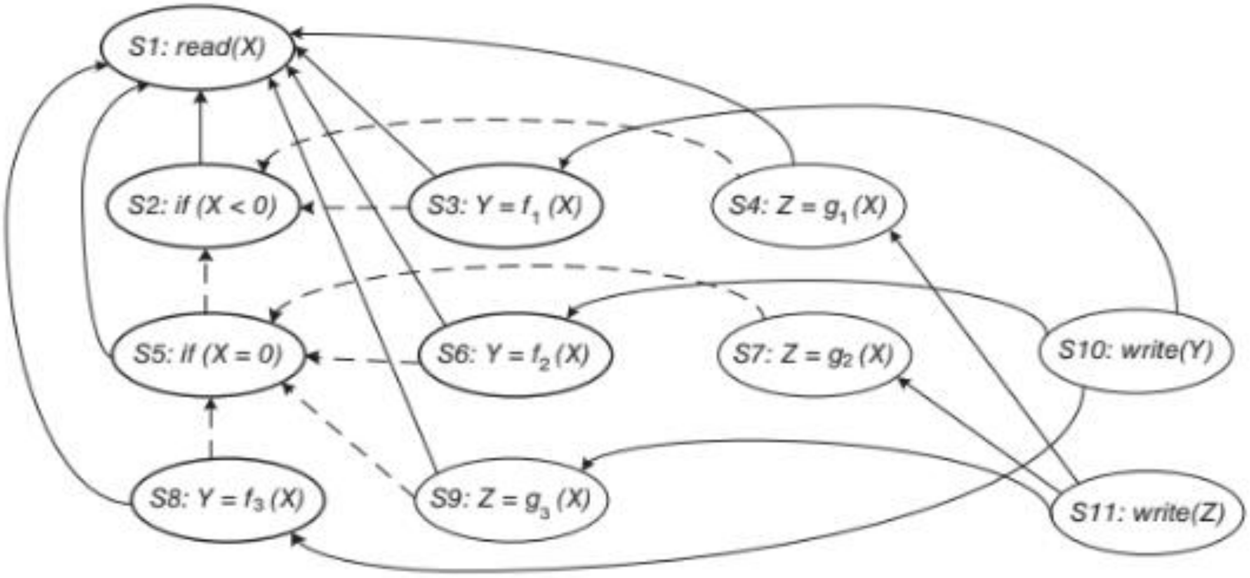

FIGURE 6.10 Program dependency graph of the program in Figure 6.9

Dependency-based Impact Analysis

Dynamic Program Slice:

Referring to the static slice example discussed above, only one of the three assignment statements, S3, S6, or S8, may be executed for any input value of X. Consider the input value −1 for the variable X. For −1 as the value of X, only S3 is executed. Therefore, with respect to variable Y at S10, the dynamic slice will contain only {S1, S2, and S3}.

For −1 as the value of X, if the value of Y is incorrect at S10, one can infer that either fi is erroneous at S3 or the “if” condition at S2 is incorrect. Thus, a dynamic slice is more useful in localizing the defect than the static slice.

A simple way to finding dynamic slices is as follows:
(i) for the current test, mark the executed nodes in the PDG; and
(ii) traverse the marked nodes in the graph.

Ripple Effect

The ripple effect shows what impact changes to software will have on the rest of the system

Stability analysis considers the total potential ripple effects rather than a specific ripple effect caused by a change.

Design stability was studied by Yau and Collofello [35] by means of an algorithm, which computes stability based on design documentation. <u>Specifically, one counts the number of assumptions made about shared global data structures and module interfaces.</u>

The key difference between design level stability and code level stability is as follows: <u>design level stability does not consider change propagations within modules.</u>

Ripple Effect

Computing Ripple Effect:

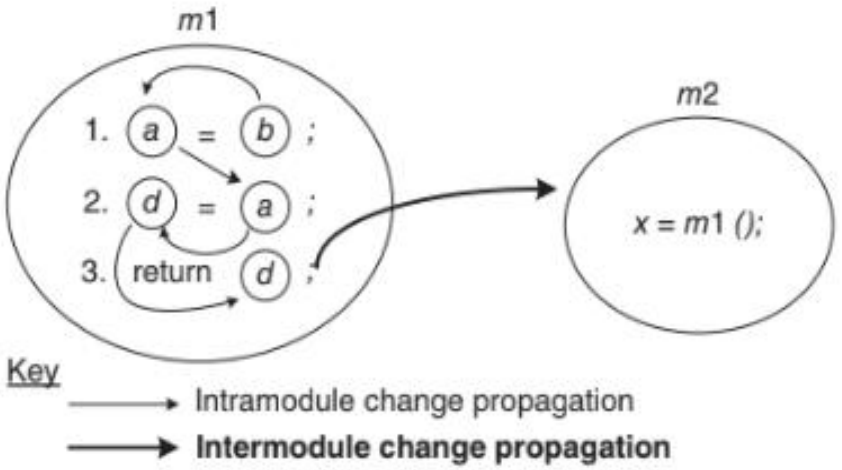

FIGURE 6.12 Intramodule and intermodule change propagation. From Reference 36.

Ripple Effect

The general expression for calculating the ripple effect for a program (REP) is as Follows

$$REP = \frac{1}{n} \sum_{m=1}^{n} \frac{V_m \cdot Z_m \cdot X_m \cdot C}{| V_m |},$$

where

1. A matrix Vm is used to represent the initial starting points for intramodule change propagation.
2. | Vm1 | represents the total number of variable definitions in m1
3. A zero–one (0–1) matrix Zm indicates values of what variables propagate to other variables in the same module.
4. For all the variables of a module m1, propaga- tion of their values to other modules is captured by an X matrix, denoted by Xm1
5. A C matrix of dimension 1 × n is chosen to represent McCabe's cyclomatic complexity, where n is the number of modules

Black, S., 2001. Computing ripple effect for software maintenance. *Journal of software maintenance and evolution: research and practice*, *13*(4), pp.263-279.

CHANGE PROPAGATION MODEL

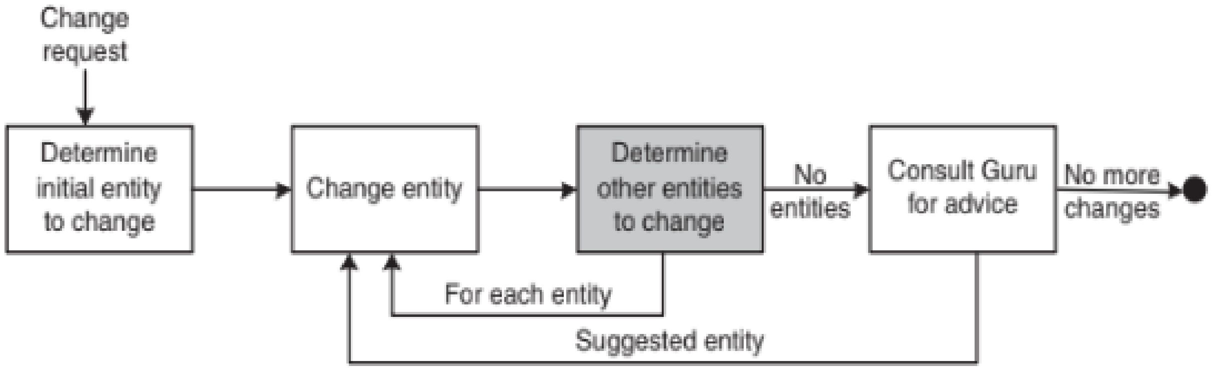

FIGURE 6.13 Change propagation model. From Reference 10. © 2004 IEEE

After receiving a change request, one identifies the initial entity in the system that needs to be changed. After changing the function, the maintainer must analyze the code to find out other, related entities to change. change. Then, those entities are actually modified to propagate the change. Similarly, the propagation process is repeated for each changed entity.

A Guru is consulted when the maintenance engineer cannot identify more entities to modify. A Guru can be a senior developer or even a comprehensive test suite.

CHANGE PROPAGATION MODEL

Gurus rarely exist and comprehensive test suites are generally incomplete in large maintenance projects.

Therefore, software maintenance engineers need good change propagation heuristics, that is, good software tools that can guide them in identifying entities to propagate a change.

The heuristic should possess a high precision attribute to be accurate and a high recall attribute to be complete.

CHANGE PROPAGATION MODEL

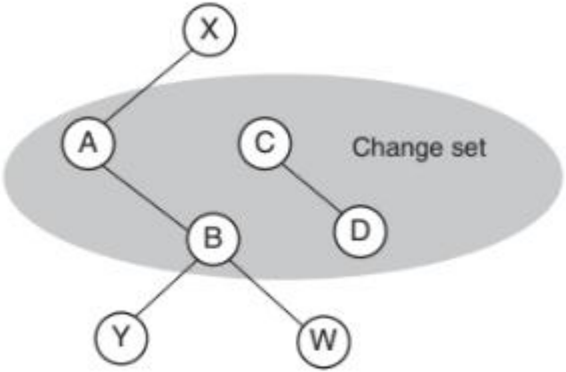

FIGURE 6.14 Change propagation flow for a simple example. From Reference 10. © 2004 IEEE

Rohan wants to enhance an existing feature of a legacy information system. He first identifies that entity A needs to be changed. After changing A, a heuristic tool is queried for suggestions, and entities B and X are suggested by the tool. Next, B is changed and he determines that X should not be changed. Now the tool is given the information that B was changed, and the tool suggests that Y and W need to be changed. However, neither Y nor W need to be changed so no changes are performed on Y and W. After having used the tool, now Rohan consults a Guru, Krushna. Krushna indicates that C should be changed. Now, Rohan modifies C and queries the heuristic for additional entities to change.

CHANGE PROPAGATION MODEL

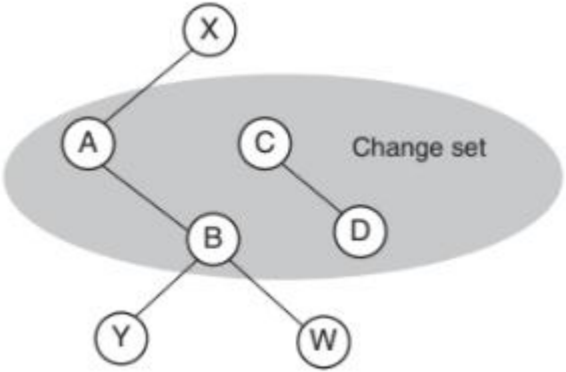

FIGURE 6.14 Change propagation flow for a simple example. From Reference 10. © 2004 IEEE

In response, D is suggested by the tool. Next, D is changed and Krushna is further queried. However, this time Krushna does not suggest any more entities for change. Now, Rohan stops changing the legacy system.

CHANGE PROPAGATION MODEL

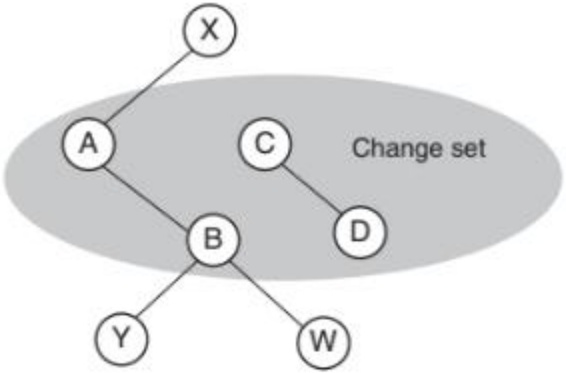

FIGURE 6.14 Change propagation flow for a simple example. From Reference 10. © 2004 IEEE

The set of entities that are changed will be called change set; change = {A, B, C, D}.

The set of entities suggested by the tool is called a predicted set. In the Rohan example, predicted = {B, X, Y, W, D}.

The entities that were required to be predicted, but were found from Guru, are put in a set called the occurred set, occurred = {B, C, D}.
That is, occurred = change − {initial entity}.

CHANGE PROPAGATION MODEL

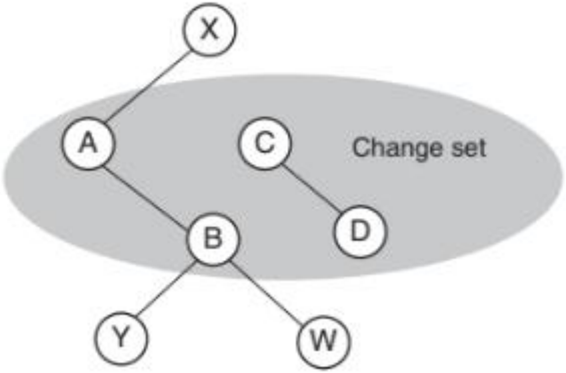

FIGURE 6.14 Change propagation flow for a simple example. From Reference 10. © 2004 IEEE

Now, recall and precision for this example are computed as follows.

$$recall = \frac{|\,predicted \cap occurred\,|}{|\,occurred\,|} = \frac{2}{3} = 66\%$$

Meaning, from the occured set, how many of them were actually predicted correctly

$$precision = \frac{|\,predicted \cap occurred\,|}{|\,predicted\,|} = \frac{2}{5} = 40\%$$

Meaning, from the total predicted set, how many of them actually occurred

CHANGE PROPAGATION MODEL

In the analysis of the above example to measure recall and precision, the authors, Hassan and Holt, made three assumptions.

- Symmetric suggestions: This assumption means that if the tool suggests entity F to be modified when it is told that entity E was changed, the tool will suggest entity E to be modified when it is told that entity F was changed. This assumption has been depicted in Figure 6.14 by means of undirected edges.

- Single entity suggestions: This assumption means that each prediction by a heuristic tool is performed by considering a single entity known to be in the change set, rather than multiple entities in the change set.

- Query the tool first: This assumption means that the maintainer (e.g., Rohan) will query the heuristic before doing so with the Guru (e.g., Krushna).

CHANGE PROPAGATION MODEL

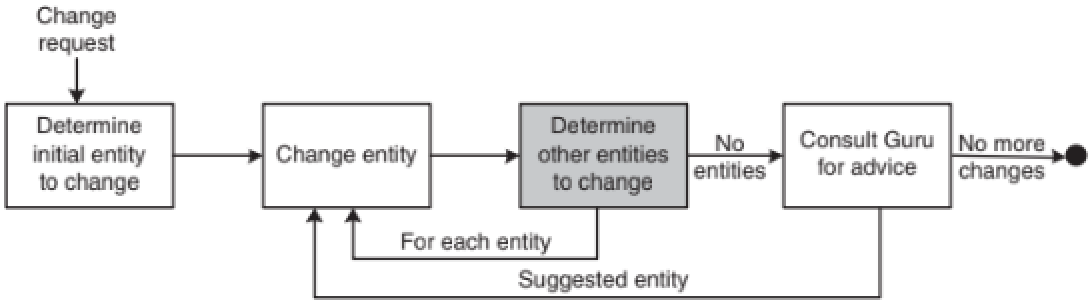

FIGURE 6.13 Change propagation model. From Reference 10. © 2004 IEEE

Heuristics for Change Propagation

The “Determine Other Entities to Change” step in Figure 6.13 is executed by means of several heuristics. The set of entities that need to be changed as a result of a changed entity is computed in the aforementioned step.

The changes can be recorded at the level of source code entities, namely, data type definitions, variables, and functions, to be able to track the following details. Modification, deletion, and addition of a source code entity. Alterations to dependencies between the changed entities and other entities in source code. For instance, it may be determined that a variable is no longer needed by a function. For each modification to the code, the corresponding modifications made to other files.

CHANGE PROPAGATION MODEL

Each heuristic discussed in this section is characterized by:
(i) data source; and
(ii) pruning technique.

Heuristic Information Sources
A heuristic can use one of many information sources to predict the entities that need to be modified.
The objectives of the heuristics are to:
(i) ensure that the entities that need to be modified are predicted; and
(ii) minimize the number of predicted entities that are not going to be modified.

CHANGE PROPAGATION MODEL

1. Entity information: In a heuristic based on entity information, a change propagates to other entities as follows.

- If two entities changed together, then the two are called a historical co-change (HIS).
- Static dependencies between two entities may occur via what is called CUD relations: call, use and define. A call relation means one function calls another function; a use relation means a variable is used by a function; and a define relation means a variable is defined in a function or it appears as a parameter in the function.
- The locations of entities with respect to subsystems, files, and classes in the source code are represented by means of a code layout (FIL) relation. Subsystems, files, and classes indicate relations between entities— generally, related entities simultaneously.

1. Developer information (DEV): In a heuristic based on developer information, a change propagates to other entities changed by the same developer. In general, programmers develop skills in specific subject matters of the system and it is more likely that they modify entities within their field of expertise.

CHANGE PROPAGATION MODEL

3. Process information: In a heuristic based on process information, change propagation depends on the development process followed. A modification to a specific entity generally causes modifications to other recently or frequently changed entities. For example, a recently changed entity may be the reason for some system-wide modifications.

4. Textual information: In a heuristic based on name similarity, changes are propagated to entities with similar names. Naming similarities indicate that there are similarities in the role of the entities.

CHANGE PROPAGATION MODEL

Pruning Techniques A heuristic may suggest a large number of entities to be changed. Several techniques can be applied to reduce the size of the suggested set, and those are called pruning techniques, as explained in the following.

- Frequency techniques identify the frequently changing, related components. The number of entities returned by these techniques are constrained by a threshold. In a Zipf distribution, a small number of entities tend to change frequently and the remaining entities change infrequently.
- Recency techniques identify entities that were recently changed, thereby supporting the intuition that modifications generally focus on related code and functionality in a particular time frame.
- Random techniques randomly choose a set of entities, up to a threshold. In the absence of no frequency or recency data, one may use this technique.

Change Management

https://www.miquido.com/blog/change-management-in-software-development/