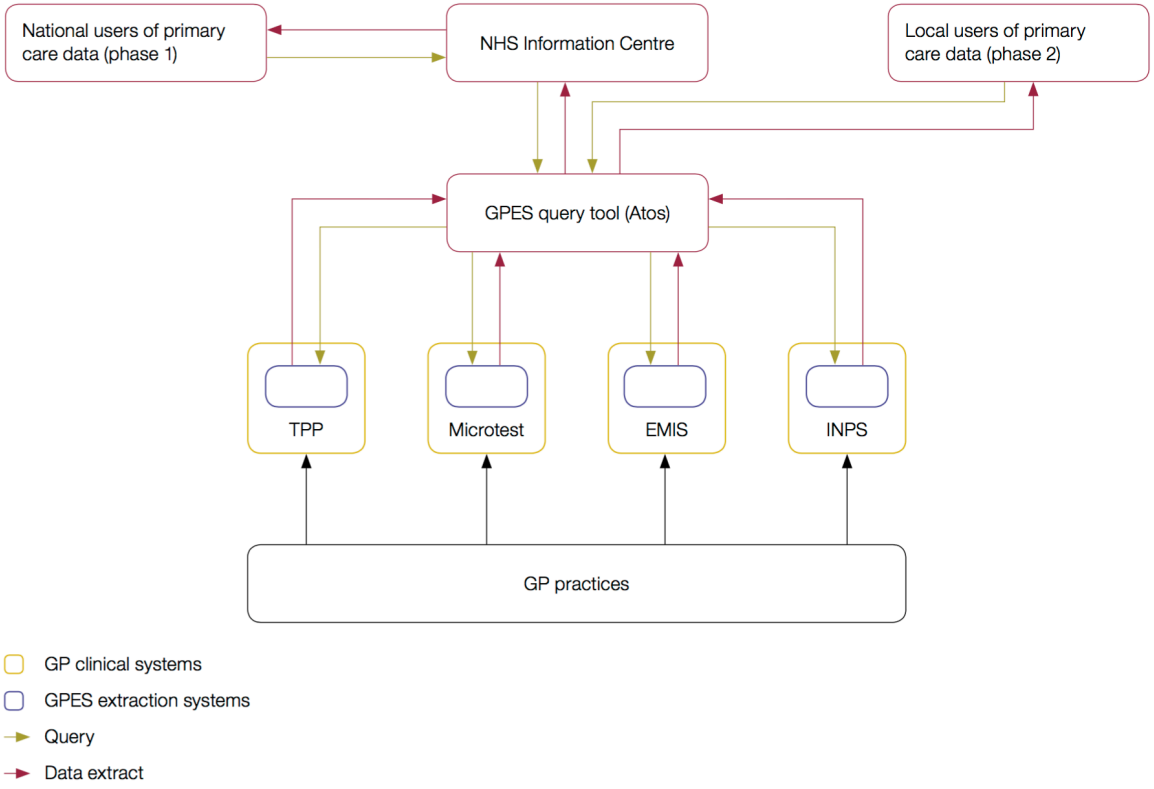
SAPM
Modifiability

Modifiability

- Just as testability is not about testing, modifiability is not about modifying – it's about how easy it might be to modify.
- This is a key area because change incurs cost.
- Four key questions:
  - What can change?
  - How likely is something to change?
  - When, where, how and by whom will changes be made?
  - What is the cost of making the change?

Look at the GPES Example

Figure 2
Design of the General Practice Extraction Service



Source: National Audit Office, based on information in the NHS IC GPES business cases

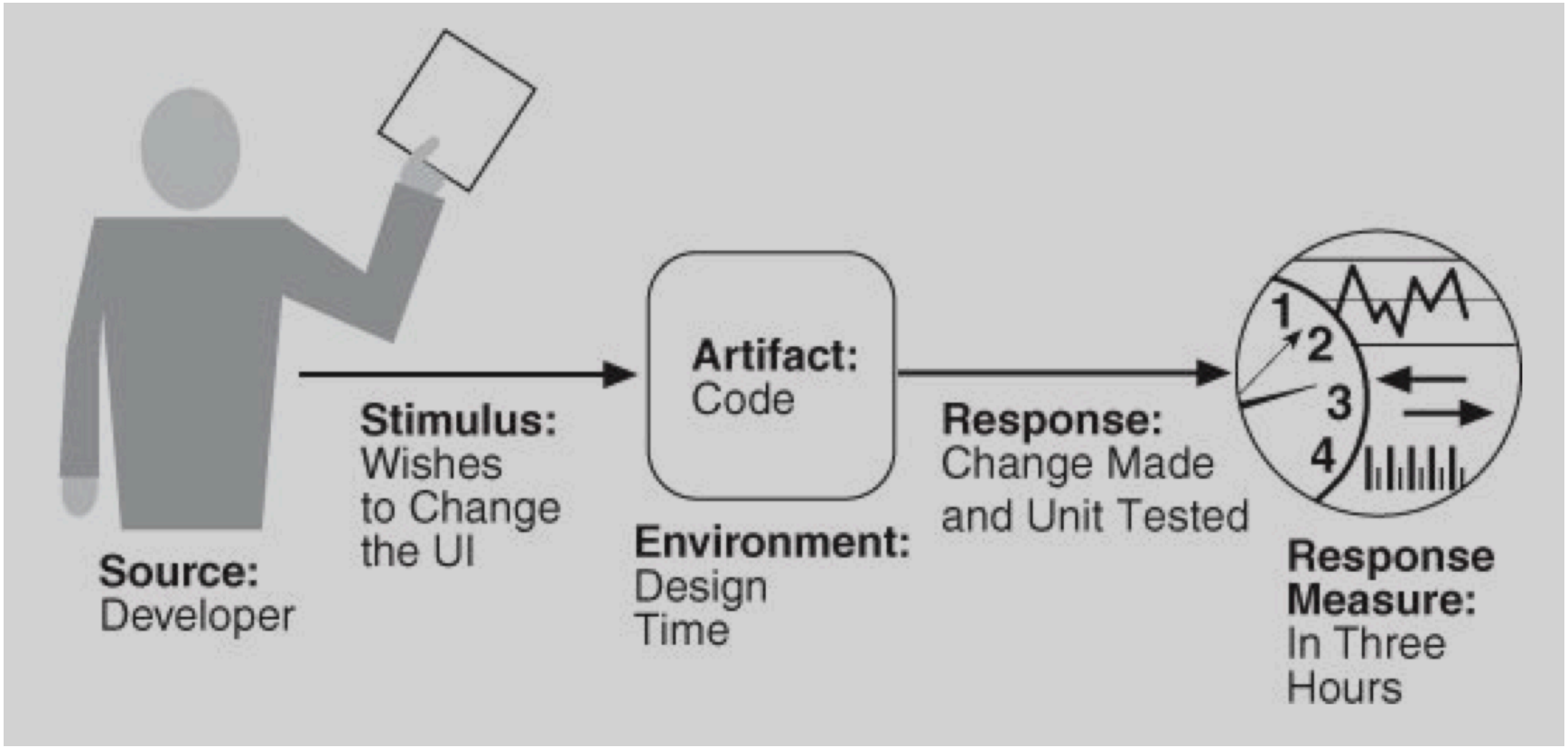
GPES: Two versions

- Version 1: General purpose query facility in each GP system.
- Version 2: Building a specific piece of business logic for each different query.
- Think about:
  - What changes can happen?
  - How likely is a change?
  - When, where, how and by whom?
  - How mush will it cost?

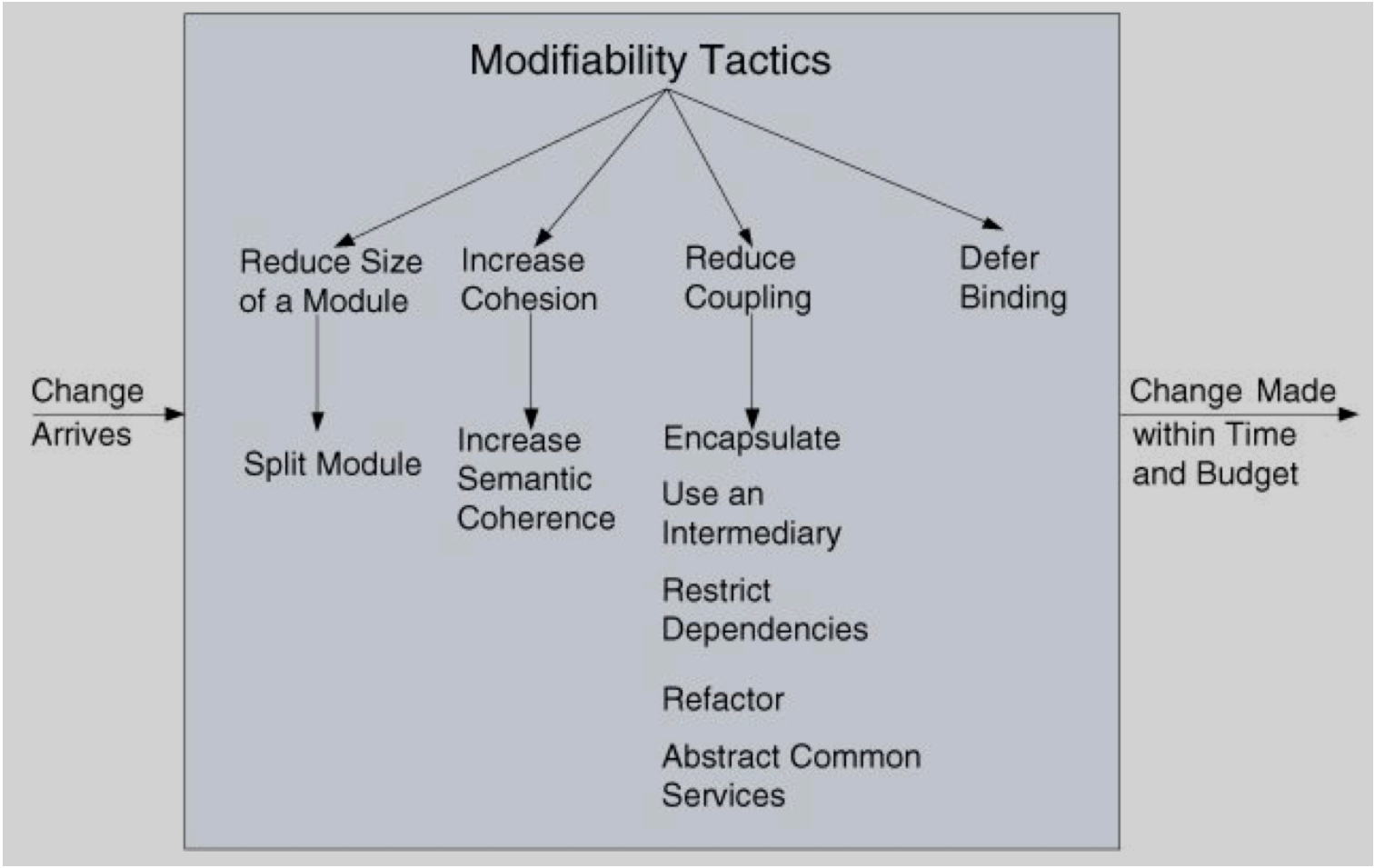
General Scenario

Portion of Scenario	Possible Values
Source	End user, developer, system administrator
Stimulus	A directive to add/delete/modify functionality, or change a quality attribute, capacity, or technology
Artifacts	Code, data, interfaces, components, resources, configurations, . . .
Environment	Runtime, compile time, build time, initiation time, design time
Response	One or more of the following: ▪ Make modification ▪ Test modification ▪ Deploy modification
Response Measure	Cost in terms of the following: ▪ Number, size, complexity of affected artifacts ▪ Effort ▪ Calendar time ▪ Money (direct outlay or opportunity cost) ▪ Extent to which this modification affects other functions or quality attributes ▪ New defects introduced

Specific Scenario



Modifiability Tactics



GPES-relevant Scenario

- **Source:** One of the stakeholders e.g. Medicines and Healthcare Products Regulatory Agency
- **Stimulus:** Wants prescribing data on NSAIDs
- **Artifacts:** Code (but depending on the architecture this could be configuration data)
- **Environment:** Operation
- **Response:** Develop the code
- **Response Measure:** Data available 5 weeks after request

GPES Version 1

- Design and validate the query with the Medicines agency.
- Code the query.
- Test on some systems to ensure it does not have bad effects.
- Rollout to all systems.
- Make the query available to Medicines agency.

GPES Version 2

- Design and validate the query with Medicines agency.
- Negotiate with the GP system providers on the design of the business logic (different in all systems?)
- Are the providers the only vendor of such services? Should it go to a procurement?
- Validate the queries on each system
- Integrate the results
- Roll out to all systems
- Make the query available to the Medicines Agency.

Modifiability Tactics

- It seems likely that the GPES V2 architecture will not pass the modifiability scenario we describe.
- Are any of the modifiability tactics appropriate to change the architecture to enable it to pass the scenario?

Modifiability Tactics

- Reduce Coupling is the category of tactics we need to consider.
- Each of the following offer potential routes with slightly different emphases:
  - Use an intermediary
  - Restrict dependencies
  - Refactor
  - Abstract common services
- “Defer Binding” means can we do this later in the process so it is more likely to be done by a computer than a human? Here this is unlikely.

More on Binding Time

- Compile time/Build Time: component replacement, compile time parameters,…
- Deployment time: configuration scripts that bind at deployment, …
- Initialization time: resource files
- Runtime: dynamic lookup, service lookup, name servers, plugins, publish-subscribe, shared repositories, (Maybe just in time compilation fits here too)

Modifiability Design Checklist

- **Allocation of responsibilities**: work out how things are likely to change e.g. technical, legal, organisational, social, markets, customers..:
  - Work out what responsibilities change.
  - Try to modularise so a change does not affect responsibilities that span many modules.
- **Coordination model**: look at how changes are likely to affect coordination and try to ensure that the most likely changes impact coordination across a small number of modules (look at GPES example)
- **Data model**: Similar to coordination model – see how a change impacts on data models and try to esnure data model changes span as few modules as possible.

Modifiability Design Checklist

- **Mapping among architectural elements**: looking at potential changes to the system, assess whether some may best be responded to by changing the mapping to elements. Explore issues such as dependencies between elements, data holdings in elements, assignment of elements to processes, threads or processors.
- **Resource Management**: Determine how a change in responsibility or quality attribute will change resource. Attempt to localise resourcing change resulting from a likely change to a small number of modules. Look at ways of using policies or configuration to manage resource change more effectively

Modifiability Design Checklist

- **Binding Time:** Control choice of binding times so there are not too many combinations to consider. Consider attempting to defer binding to later, balance this against the cost of providing a later binding mechanism. (e.g. GPES example again).
- **Choice of Technology:** Choose technologies that make the most likely changes easier (e.g. choose a technology that allows runtime alteration of critical parameters rather than one where parameters are chosen at compile time) but balance this agains the cost of the different technologies.

Summary

- The standard approaches look to improve cohesion and reduce coupling in the structure in all of the views (static, dynamic, deployment).
- Costs need to be taken into account, making something easier to change later in the process may be useful but excessively expensive.
- Often introducing mechanisms that allow us to make changes late in the development cycle helps ease modifiability costs.