**Dynamically Adjustable Software Process**

by

Yasmeen Rawajfih

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
December 10, 2016

Keywords: Software process tailoring, Software Process Improvement (SPI), Software process evaluation, Agile software process



Doctoral Committee

David Umphress, Chair, Professor of Computer Science and Software Engineering
James Cross, Professor of Computer Science and Software Engineering
Dean Hendrix, Associate Professor of Computer Science and Software Engineering
Wei-Shinn Ku, Associate Professor of Computer Science and Software Engineering

Abstract

Industry experience shows that the quality of a software product is directly related to the quality of the software process used to produce it. Software process can rarely be used out of the box. Project types, scopes and developer skills differ, making it crucial for a process to be tailored before it can be applied successfully to a software project. Heavy-weight software processes are typically tailored after the conclusion of a project focusing benefit on future projects, whereas agile software process tailoring benefits the current project but lacks an orderly approach, relying instead on intuition and hunches. What is missing is a middle ground in which software practices are adjusted while a project is underway and are adjusted in a disciplined fashion.

This research introduces an engineered approach to software process adjustment based on a software developer's "pain points". This tailoring method is responsive (happens in time to affect the near term), systematic (has a defined process), and is based on evidence (ties back to best practices in industry). The Dynamically Adjustable Software Process (DASP) framework was built to assist software developers in tailoring their software processes based on pain points. After determining the source of pain it evaluates alternative software practices and provides the developer with a recommendation that best achieves his/her process goals and best fits the project's needs.

The concept of tailoring process based on a developer's pain points was validated in a real-world context by performing a case study in which software engineering students used this method to tailor the software process they were using. Results show that a majority of students said this tailoring method improved their adherence to process and was in line with their personal style of software development. A majority also indicated they would adopt this method of tailoring in the future.

Acknowledgments

This dissertation could not have been completed without the help and support of many individuals. I would like to convey my heartfelt gratitude and sincere appreciation to all the people who have helped and encouraged me along the way.

First and foremost, I would like to express my deep appreciation and gratitude to my advisor Dr. David Umphress, who has been an outstanding educator, mentor, and inspiration to me throughout both my master's and PhD degrees. I truly appreciate all the time, ideas, and expertise you have contributed to make this work possible. Thank you for your continuous insight, guidance, encouragement, and sense of humor, and for always being patient and flexible. You have made this an exciting and rewarding journey.

I would like to thank my dissertation committee members Dr. James Cross, Dr. Dean Hendrix, and Dr. Wei-Shinn Jeff Ku. I am grateful for all of your support, guidance, and valuable input. I would also like to thank Dr. Jorge Valenzuela for taking time out from his busy schedule to serve as my external reader.

I would like to thank my dear friend and research companion Haneen AlAbdulRazzaq to whom I am eternally grateful. Without her encouragement and support I would not have been able to complete this journey. Thank you for helping me stay on track, believing in me, and always being eager to listen and offer help and advice. Thank you for making our time in the lab fun and productive and for all the good times and memories both on and off campus. I would also like to thank my dear friend Ahmad Alsmair for his continuous insight, encouragement, and support. Thank you for always offering a fresh perspective and for the many hours you spent proof reading.

Finally, I would like to thank my parents Linda and Zahir Rawajfih, my brother Nabeel, and my sister Hadiah for their unwavering love and support. Thank you for always believing in me and encouraging me in all my pursuits.

Table of Contents

List of Figures

List of Tables

# Chapter 1

# Introduction

Brooks [Brooks, 1987] notes that building software is an inherently difficult endeavor, one that is complicated further by the languages, tools, and methods we employ. This artificially-induced complexity is reduced as software development techniques evolve, but the evolution mechanism is, itself, fraught with both intrinsic and manufactured difficulties. At issue is how to adjust the software development process so that it is responsive, effective, and systematic. The conventional school of thought suggests that the point at which a software development process should be tuned to minimize complexity is when enough project evidence has been collected to substantiate change. The contemporary agile philosophy is to alter a software development process as soon as it is perceived to be dysfunctional. The former typically adjusts software practices at the conclusion of a project in preparation for the next, focusing benefit on future projects; the latter benefits the current project, but lacks an orderly approach, relying, instead on intuition and hunches. What is missing is a middle ground in which software practices are adjusted while a project is underway and are adjusted in a disciplined fashion.

## 1.1 Software Process

Software process emerged as software began finding its way into every discipline, and increasingly became an essential part of almost every aspect of our daily activities. This introduced the need for larger more complex pieces of software that exhibited an array of non-functional requirements such as scalability, reliability, and maintainability. Companies that offered software services found themselves having to meet strict quality standards and rigorous deadlines in order to survive the competitive industry. The need for software process and a means

to engineer the development of software in a more structured and systematic fashion while staying on schedule drove companies to adopt software process and researchers to delve into the different aspects of software process. The initial introduction of software process was in the 60's and 70's in the form of software lifecycles. Software lifecycles provided guidelines for the stages of software development and presented a philosophy, but lacked the specifics on how to carry out these stages. In the 80's software process became a field of its own and gained popularity as workshops, journals, and organizations emerged to support, develop, and promote software process [Fuggetta, 2000]. As process became more commonly adopted, standards were introduced and companies were often encouraged to meet those standards in order to qualify for government contracts or projects of a certain magnitude or caliber. For a while it seemed that the more structured and heavy-weight a process was, the more it was perceived as a valid and dependable way to develop software. This opinion was formed primarily by higher management and stakeholders, but not necessarily by the developers and other contributors that implemented and carried out all the strenuous details of the process. As the software industry expanded and software projects grew larger and more competitive, changes in requirements became inevitable. The Agile Manifesto for Software Development [Beck et al., 2001], introduced in 2001, supports changes in requirements and "individuals and interactions over processes and tools", in addition to frequent delivery of working software and better communication between the stakeholders of a project. It also calls for teams to meet at regular intervals to reflect on how to become more effective and adjust their behavior accordingly [Beck et al., 2001]. With the introduction of agile development, many existing software processes that had previously been labeled as "lightweight", such as SCRUM and XP, were considered the de facto standard for agile [Banerjee, 2012] [Alliance, 2013]. Many enthusiasts adopted the agile methodologies, and agile gained a large following in the development community [Liddle, 2013] [Highsmith, 2002]. This allowed developers to experience the benefits of a light-weight software process and its significance in the successful completion of software projects, while spending less time on documentation and formalities.

Agile may have solved many problems that heavier weight processes brought about, but faced some challenges of its own such as scalability, tool support, credibility, etc [Hafterson, 2011].

## 1.2 The Challenges of Process: Process Tailoring

Software process tailoring, also referred to as Software Process Improvement (SPI), has been gaining popularity and support for many years since industry experience shows that the quality of a software product is directly related to the quality of the software process used to produce it [Humphrey, 1995]. Software process can rarely be used out of the box [Kalus and Kuhrmann, 2013]. Project types, scopes, and teams differ, making it crucial for a process to be customized before it can be applied successfully to a software development project. Process granularity often needs to be customized based on the degree of formality and desired outputs of the process being used, making tailoring one of the major challenges of successfully implementing a software process. Process tailoring itself faces its own set of challenges. It is difficult to get developers to acknowledge the shortcomings in their current ways of working [Wiegers, 2005], which makes it difficult to identify the actual problem areas in the process that need to be addressed. The lack of commitment from developers, and insufficient time to learn and adopt new techniques also impacts tailoring a software process [Gallivan, 2003][Janes et al., 2008]. If results cannot be immediately observed, enthusiasm often dwindles before adjustments to the process have been correctly implemented [Wiegers, 2005].

## 1.3 Conventional Post-project Approach

Tailoring a heavy-weight software process typically occurs after the completion of a development effort, or at regular intervals determined by the organization. This delay in tailoring means that even if a problem in the process is identified, it is not addressed until the end of the project, which may be months or even years. Delaying the improvement of a software

process would potentially lower the quality of the software product or increase the cost of development. If software processes were tailored mid-project and problems were addressed immediately, this could improve the quality of the software being developed and lower the costs involved. Having the ability to dynamically tailor a software process based on what is causing the most pain or what will produce the largest value with the least amount of change to the process would be extremely valuable. If a software process existed where developers would be able to use the minimum amount of process practices that provide the largest payoff, and immediate dynamic tailoring was possible based on their pain points, developers might actually embrace this software process and their attitude toward process could change.

### 1.4 Agile In-Situ Approach

Agile approaches advocate "fixing" the process when it is "broken" but do not prescribe how to carry this out in a systematic fashion. [Beck and Andres, 2004] propose continuously improving software process by doing "the best you can today, striving for the awareness and understanding necessary to do better tomorrow". They suggest improvement through trial and error, and state that "the course of improvement is not smooth or predictable" but rather context sensitive. [Conboy and Fitzgerald, 2007] state that "once you get comfortable with the basic process, you will grow it to fit your situation more precisely", but do not offer any detail on how this can be achieved.

### 1.5 An Engineered Approach to Software Tailoring

Developers show apprehension toward adopting any new software process, including agile processes, mainly because they believe some of the activities are unnecessary, and adopting a new set of activities usually has a higher adaption curve than they are comfortable with. Allowing developers to use a customized personal software process with the least possible number of necessary activities, and the ability to dynamically make adjustments to that process as soon as a problem surfaces, may be just what developers need to finally and truly

embrace personal software process. [Moczar, 2013] states that "dynamism means the ability to switch strategies when the current one isn't working". To achieve this, an "engineered" approach to software process adjustment that is responsive (meaning, happens in time to affect the near term), systematic (meaning, has a defined process), and is based on evidence (meaning, ties back to best practices in industry) is needed.

# Chapter 2

# Literature Survey

## 2.1 Tailoring Software Process

Following is an overview of how software process tailoring has been addressed in software processes of different weight and scale. Examples of efforts to create customized processes and tailor them to a specific project's needs are also discussed.

### 2.1.1 Tailoring in Heavyweight Software Process

#### 2.1.1.1 CMMI

The Capability Maturity Model Integrated (CMMI) is a framework of best practices that provides guidelines for the construction and management of software development products and processes. It is also a process assessment framework that offers evaluation criteria for an organization's process maturity level [CMMI, 2001]. CMMI was developed by the Software Engineering Institute (SEI) at Carnegie Melon University, and was the first assessment methodology supported by the government to offer a means of evaluating software companies contracting with government organizations [Kuvaja, 1994] to guarantee a certain quality level of the software being produced. CMMI does not prescribe the specific tasks and activities to be carried out in a software process, but rather the characteristics that should exist in an organization's set of processes and activities. Tailoring a software process in CMMI is a formal activity in which an organization's process is tailored prior to the beginning of a project or "organizational function". The high-level tailoring CMMI describes does not specify practices to use, however, it provides detailed steps regarding the manner in which a process should be tailored, and lists a number of subpractices that should be followed such

as documenting the defined process and keeping records of any tailoring that occurs. In CMMI, an organization's "standard process" is the starting point for tailoring, from which the relevant processes and life-cycle elements are selected and tailored to address the needs of a specific project. The resulting process is referred to as the "defined process" and is then monitored and objectively evaluated to ensure that the deliverables and services it provides meet the required objectives and standards. [CMMI, 2001] states that "the organization's process objectives should be appropriately addressed in the defined process". The results of the process are reviewed by a team members and higher-level management in order to take corrective actions. [CMMI, 2001] instructs to "revise the description of the defined process as necessary" but does not offer any insight into when or how often this should occur. Tailoring in CMMI can include modifying a life-cycle model, combining elements of different lifecycle models, modifying process elements, replacing process elements, and reordering process elements [CMMI, 2001]. Tailoring of a process may occur to address issues specific to a project such as the customer, cost, schedule, quality tradeoffs, technical difficulty of the work, and experience of the people implementing the process [CMMI, 2001]. In addition, each organization should have a set of documented "tailoring guidelines" which specify how the organization's standard process can be tailored. The tailoring guidelines themselves are submitted for peer review and can be revised as necessary. The organization's process documentation standards are specified in these guidelines, in addition to process deviation procedures. Deviations from a defined process must be submitted for approval and approved before they can take place.

#### 2.1.1.2 ISO 9000 Series

The ISO 9000 series of standards was developed by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) to "provide guidance and tools for companies and organizations who want to ensure that their products and services consistently meet customer's requirements, and that quality is consistently

improved" [ISO, 2015]. As ISO 9001:2000 is a general standard created to apply to any organization regardless of the type of system it employs, many interpretations in the form of supporting documents have been created to provide guidance for organizations of specific types that wish to comply with the ISO standard.

**ISO 90003:2004**

ISO 90003:2004 provides guidelines for organizations to apply ISO 9001:2000 to the computer software field [ISO/IEC, 2008]. ISO 90003 states that an organization must have a software process (life cycle model) in place but does not specify any required criteria for selecting the process to use, other than that it should be suitable for the nature of the project. Tailoring a process in ISO 90003 is a continuous effort that occurs as needed as the project progresses. ISO 90003 does not specify when to tailor, but does require organizations to tailor their processes when the need arises. Data must be collected and analyzed in order to support effective management of the processes. Tailoring is considered part of "quality planning" in ISO 90003.

#### 2.1.1.3 Bootstrap (Developed 1990-1993)

Bootstrap is a process improvement and assessment framework that was developed as part of the ESPRIT project with the unique characteristics and demands of the European software industry in mind [Kuvaja, 1994]. Bootstrap was influenced by SEI's CMM and the ISO 9000 standard and aimed to "set practical standards with no particular geographical area, industrial sector, or timing constraints" [Kuvaja, 1994]. Bootstrap's view on process tailoring is that it is an "industrial production process" that follows the Kaizen/Shewhart iterative cycle of plan, do, check, and act [Kuvaja, 1994]. Bootstrap's continuous improvement philosophy was adopted from the Total Quality Management (TQM) strategy in which continuous improvement is required to provide an organization with a competitive advantage in the market and better meet customer expectations [Kuvaja, 1994]. Process is considered

a "temporal entity" which accepts requests and outputs results. The performance of a process is monitored against predetermined goals and feedback is continuously fed back into the process making it a "continuously self-correcting" process [Kuvaja, 1994]. As can be observed in other process assessment frameworks, Bootstrap is "non prescriptive" in the sense that it offers high-level guidelines to process improvement without specifying the means to achieve them [Kuvaja, 1994]. Bootstrap states that the improvement activities are repeated whenever a problem arises, but does not specify what constitutes a problem in the process. Improvement must be planned in detail to generate an "action plan", and the process of improvement itself is considered a small project that requires the allocation of various resources such as time, budget, and quality management. Improvement of an organization's software process can be motivated by the frequently evolving industry challenges or in order to better meet an organization's business goals. Improvement elements in Bootstrap include an improvement strategy, improvement model, organizational sponsorship, and an improvement process. Bootstrap follows an "assessment-oriented" process improvement approach, meaning it considers process assessment a prerequisite to process improvement [Kuvaja, 1994]. The assessment results are used to build the improvement strategy, and different levels of goals are defined. The improvement model is also determined by the assessment results and the process improvement goals [Kuvaja, 1994]. Improvement in Bootstrap covers the "Organization, Methodology, Technology" criteria specified in the ISO 9000 standard, where organization refers to the quality management and process evaluation criteria, methodology refers to the actual procedures in place, and technology refers to the tools being used to achieve the process quality goals.

#### 2.1.1.4 ISO/IEC 15504 (SPICE) (Developed 1992-1998)

Similar to SEI's CMMI, the Software Process Improvement and Capability dEtermination (SPICE) model is an international standard model that was developed as an attempt "to create a way of measuring process capability, while avoiding a specific approach to improvement

such as the CMM's maturity levels" [Konrad et al., 1995]. It "defines at a high level, the fundamental objectives that are essential to good software engineering" [ISO/IEC, 1998a]. SPICE provides guidelines for continuous process assessment and improvement but does not specify how to achieve these goals. In addition to offering a large collection of best practices that should be implemented in an organization's processes, SPICE is a process capability reference model that allows for the evaluation of an organization's process maturity. With regard to process assessment, SPICE offers an evaluation of each process in an organization independent of other processes, whereas CMMI evaluates all processes in place as a unit. Similar to Bootstrap, SPICE refers to process tailoring as process "improvement". Process tailoring in SPICE is a formal process in which various organization members participate. Tailoring must be initiated and supervised by higher management, but is considered a continuous team effort that requires the investment of time and other resources[ISO/IEC, 1998b]. Tailoring occurs to address business goals or other defined needs of an organization, and requires the organization to specify its measurable improvement goals and prioritize them before an improvement effort is launched. Process improvement in SPICE is considered a project in itself, and the improvement plan must be assessed and analyzed before improvements are prioritized for implementation. Improvement efforts are monitored and compared against the improvement plan to ensure compliance and that the improvement goals are being met. SPICE indicates that processes and their practices must be continuously monitored and evaluated quantitatively to ensure their effectiveness and efficiency in achieving the organization's needs and business goals. SPICE does not offer any detail on what metrics should be used but rather leaves that responsibility to an organization to determine based on the industry's current software engineering best practices and the specific needs and goals of that organization. Similar to CMMI's tailoring guidelines, SPICE also states than an organization should have an improvement process with well-defined activities that is reviewed regularly, and that "appropriate actions should be taken where any discrepancies have been identified" [ISO/IEC, 1998b].

### 2.1.2 Tailoring in Agile Software Process

#### 2.1.2.1 Extreme Programming (XP)

Extreme Programming (XP) gained its name from the concept of taking "best practices" to an extreme level [Wikipedia, 2015a]. XP was created by Kent Beck during his work on the Chrysler Comprehensive Compensation System in 1996 [Wikipedia, 2015a], and was first introduced in [Beck, 2000]'s text titled "Extreme Programming Explained: Embrace Change" and later refined in the 2nd edition [Beck and Andres, 2004], also referred to as XP2 [Dudziak, 1999][Mnkandla, 2008]. XP is an incremental iterative process that promotes frequent releases, continuous integration, and refactoring [Dudziak, 1999]. XP also prescribes a number of practices such as pair programming, test driven design, following coding standards, the "planning game", user stories, standup meetings, collective ownership of the code, and forty-hour weeks [Dudziak, 1999][Meso and Jain, 2006]. XP claims that four interrelated control variables affect software projects: cost, time, quality, and scope [Dudziak, 1999]. XP is also centered around four core values: communication, simplicity, feedback, and courage. Continuous and open communication between all stakeholders plays an essential role in any XP project. Communication can take many forms in XP including direct communication and documentation [Dudziak, 1999]. As declared in one of its design principles, XP follows the concept of "do the simplest thing that could possibly work". Feedback from all stakeholders and at all stages of a project plays a major role in XP projects. The sooner feedback is received, the easier and less costly it is to take corrective actions. Courage refers to aggressiveness in carrying out the "extreme" practices of XP as many of them contradict those of traditional software process [Dudziak, 1999]. XP's two major design rules are "Do The Simplest Thing That Could Possibly Work" (DTSTTCPW), and "You Are Not Gonna Need It" (YAGNI), which states that developers should not implement features that will possibly be needed in the future because they will consume time and require maintenance and will possibly never be required. Although tailoring is not explicitly mentioned in XP,

since continuous communication and feedback are key, it is assumed that any process-related issues that arise will be addressed during these sessions and corrected as soon as possible to reduce cost. For this reason it can be assumed that any tailoring that occurs will take place in an ad hoc fashion.

#### 2.1.2.2 Scrum

Scrum was developed by Ken Schwaber and Jeff Sutherland in the early 1990s to help organizations struggling with complex development projects [Schwaber, 2004]. Scrum is an iterative incremental software development process designed to adapt to continuous change in requirements [Wikipedia, 2015c]. Unlike XP, Scrum does not specify any development practices to be carried out during implementation, but rather focuses on the management aspect of software process and the interactions between team members to support the ideas of flexibility, adaptability, and productivity [Abrahamsson, 2002]. Roles play an important part in the Scrum process. Scrum [Abrahamsson, 2002] identifies six major roles that contribute to the success of the process: The Scrum Team, Scrum Master, Product Owner, Manager, User, and Customer. Sprints are iteration lengths or units of time typically equal to thirty calendar days, in which the goals set for an iteration are implemented and an "executable increment" is built [Abrahamsson, 2002]. Artifacts used in Scrum include a continuously evolving Product Backlog to manage requirements, a Sprint Backlog to manage tasks assigned to the next Sprint, a Sprint Planning Meeting in which all stakeholders agree on the functionality to be added to the next sprint, a Daily Scrum Meeting to track the progress of the sprint and address any issues that surfaced since the last meeting, and Sprint Review Meeting where all stakeholders are updated with the results of the sprint. Process tailoring in Scrum occurs in an ad hoc manner during the Daily Scrum Meetings where any deficiencies in the process or practices used are identified and addressed. Since these meetings are conducted by the Scrum Master, s/he is responsible for approving any modifications to the process.

#### 2.1.2.3 Adaptive Software Development (ASD)

Adaptive Software Development (ASD) emerged to support "extreme projects in which high-speed, high-change, and uncertainty reign" [Highsmith, 2000]. Rather than the static cycle of plan, deliver, review, ASD proposes a dynamic cycle of speculate, collaborate, learn, that promotes continuous learning, reevaluation, and adjustment [Highsmith, 2000]. Similar to Scrum, ASD focuses on the management aspect of software process and does not require specific practices to be used as long as they support the process' philosophy. [Abrahamsson, 2002] notes that "ASD is the most abstract method from the software development viewpoint" [Abrahamsson, 2002]. In [Highsmith, 2000], Highsmith defines the six characteristics of adaptive lifecycles to be: mission focused, component based, iterative, timeboxed, risk driven, and change tolerant. A mission and timebox are defined for each project when it is initiated, then the number of cycles and timebox for each cycle (typically four to eight weeks) are identified. As in Scrum, each cycle in ASD must deliver working components that are integrated into the existing product. As ASD is risk-driven, the components selected for each cycle are the highest risk components. Tailoring in ASD occurs during the "learn" component of ASD's lifecycle. A feedback process called the "people-and-process review" [Highsmith, 2000], occurs at the end of each cycle (called mini-postmortems) as well as the end of the project (called a postmortem). Four general questions are suggested as a means of evaluation: what's working?, what's not working?, what do we need to do more of?, and what do we need to do less of?. These questions are recommended to help teams learn more about themselves and the process being used, but leave the corrective actions up to the team to decide.

#### 2.1.2.4 Feature Driven Development (FDD)

Feature Driven Development (FDD) is an iterative and incremental agile software process that was introduced by Palmer and Felsing in 2002 [Mnkandla, 2008]. FDD is a supportive process that only addresses the design and implementation stages of a lifecycle and needs to

be combined with other agile processes to provide a comprehensive process lifecycle [Abrahamsson, 2002]. As its name indicates, FDD focuses on the features, defined as client-valued functionality [Goyal, 2008], of a development effort and provides guidance on identifying, designing, and implementing these features [Abrahamsson, 2002]. Similar to Scrum and ASD, FDD provides frequent releases of working components. It emphasizes high quality, and claims to be suitable for building critical systems as well as the ability to support larger teams than most agile processes [Goyal, 2008]. FDD is composed of five stages: Develop an overall model, build a feature list, plan by feature, design by feature, and build by feature. In order to estimate progress, FDD has six milestones that are used for the Design By Feature (DBF) and Build By Feature (BBF) stages: domain walkthrough, design and design inspection are performed for the former, whereas code, code inspection, and promote to build are used for the latter. Each of the milestones is assigned a weight represented as a percentage of the completion of the feature [Goyal, 2008]. Features are tracked in a feature set, containing the number of features and their completion status to provide stakeholders with a visual representation of weekly progress. FDD defines six major roles: Project Manager, Chief Architect, Development Manager, Chief Programmers, Class Owners, and Domain Experts, in addition to a number of supporting roles. Tailoring in FDD is performed through metamodeling, which contains both business and technical activities and provides insight into the practices and subpractices that are being performed to achieve the five stages of FDD [Wikipedia, 2015b]. Metamodeling is consistent with the UML diagrams that FDD makes use of in its design and development stages. FDD does not specify when to tailor the process, but it is assumed that issues related to both the product and process are addressed during the inspections that occur regularly.

### 2.1.3 Automated Attempts to Create and Tailor Custom Software Process

#### 2.1.3.1 The Generic Agile Methodologies (GAM) Framework

Mnkandla [Mnkandla, 2008] suggests customizing software process by selecting specific practices from multiple agile methodologies rather than adopting an entire agile methodology as is. The GAM method was developed to help guide software firms to finding the most project-appropriate agile practices to use based on the nature of software projects at hand and tuning the practices to fit the project's requirements and development environment. Mnkandla's [Mnkandla, 2008] framework is composed of three stages: classifying projects for the suitability of agile methodologies, determining the agile methodologies appropriate for a project and selecting specific practices from these methodologies to be used, and finally tuning the selected practices in a novel technique called the GAM model. The GAM model tries to fill the gaps in the existing documentations of agile methodologies that do not state the way in which a methodology can be tailored based on the project environment [Mnkandla, 2008] [Abrahamsson et al., 2003]. The author categorizes practices of software processes into two types: "Iterative Incremental Development" referring to the lighter-weight, more agile practices, and "Plan-Driven Development" referring to heavier weight practices borrowed from heavier-weight methodologies. [Mnkandla, 2008] states that projects many a time require the use of practices from both groups, based on the project's requirements and nature. The philosophy behind the framework is that all activities of a process "should be driven by what the customer values and prioritizes in a project" [Mnkandla, 2008]. Furthermore, any practice in a development process should be classified into either a social practice or a technical practice. For the first of the three stages in the framework, [Mnkandla, 2008] developed a matrix to classify projects and determine their suitability for the use of agile methodologies. The matrix was developed using semi structured interviews with project managers and software developers to find the most important parameters affecting software development and software process. The data collected was compared with Boehm and Turner's [Boehm

and Turner, 2003] risk-based approach to create the matrix. In the matrix, five project parameters are each given a rating from 0 to 5 (zero being the most agile and 5 being the most rigorous) in order to determine the suitability of agile development for that project. A second matrix is used in the second stage of the framework in order to find the agile methodologies most relevant to the project at hand and select an appropriate set of agile practices from these agile methodologies. The second stage matrix was created by combining the approaches of [Avison and Fitzgerald, 1995], [Abrahamsson, 2002], and [Boehm and Turner, 2003]. The second stage of the framework involves two steps: the first step is to determine whether the project's requirements map to one or more agile methodologies. If they map to one, then that agile methodology should be used, whereas if they map to more than one, then creating a matrix is require. A table must be created for each methodology that is relevant, with a list of parameters and a rating on a scale from 0 to 5 (zero being full satisfaction and five being complete variance) on how well the description of each parameter satisfies the customer's values and priorities. The scores from each methodology's parameters are summed up, and a matrix is created containing all relevant methodologies and their scores for each parameter. Practices are then selected from the different methodologies based on low scoring (highly satisfying customer needs) parameters. The third stage of the framework tunes the selected practices to better fit the project at hand, using a novel approach called the GAM model. The GAM model itself is comprised of three stages: the first phase identifies social practices, the second phase identifies technical practices, and the third phase identifies the practical tuning steps required [Mnkandla, 2008]. Despite the fact that the GAM framework offers very detailed information on how to select agile practices and fine tune them to fit a specific project, it does not offer any details on how to tailor the resulting process once it has been created. Furthermore, even though [Mnkandla, 2008] mentions the importance of combining agile and non-agile methodologies in some cases, only agile methodologies are supported in their work and including non-agile methodologies is listed as possible future

work. A standalone computer application named the "GAM Toolbox" is also suggested as future work to help find a set of recommended agile practices suitable for a specific project.

#### 2.1.3.2 Building an Organizational Repository of Experiences (BORE)

Henninger [Henninger, 2001] proposes a methodology and supporting web-based tool to aid software organizations in capturing project experiences and dynamically tailoring their Standard Development Methodologies (SDMs) to fit individual project needs. They state that "SDMs must become living documents that evolve with changing software development needs." [Henninger, 2001]. BORE (Building an Organizational Repository of Experiences) is a prototype built to aid in maintaining and customizing the SDM of an organization and recording new experiences in a centralized knowledgebase that can be used to benefit future projects. As [Henninger, 2001] points out, BORE "does not define a specific development methodology, but provides tools to create and refine organization-specific methodologies." BORE initially contains a defined SDM, which is a general defined process that outlines the activities and steps that should be followed when an organization builds software. As the BORE repository is used, and organization-specific knowledge is added, it becomes easier to use and contains data that is more relevant to the organization's needs. Multiple project domains are built into BORE, each representing a separate SDM, and additional domains can be added as each organization builds and customizes its own version of BORE. Domains are considered different paths a project can take, and each contains activities and rules that are required by projects that belong to that particular domain. Each project must belong to a single domain, which determines the rules that apply and activities that will be added to the project's list of required activities as a consequence. Rules are used to specify when the activities should be carried out. Each activity can have multiple cases attached to it in which teams document their progress and from which they can draw on previous project experiences. After a project domain is selected and its associated rules and activities are added to the project's process, the process is tailored to fit the project requirements. Tailoring in

BORE occurs by matching project characteristics to rules. For each rule selected, the associated activities that need to be performed will be added to the process. Different activity options can be selected to further refine the tailored process. Both rules and activities can be edited through the web-based tool to better match a project's specific needs. The tailored process is reviewed periodically in order to make any changes necessary. If it is determined that the BORE options that were originally selected were incorrect or new information surfaced that caused the selected options to require change, new BORE activities are selected and customized by selecting the applicable rules and their corresponding activities. If it is determined that the existing rules and activities available in the repository are insufficient, a deviation request is generated in order to add new rules and activities to the repository. After the process has been tailored, changes to the process are reviewed carefully to ensure that correct actions have been taken to improve the process before the tailoring is approved. Before any added rules or activities are added to the domain's SDM, the deviations must be reviewed carefully and modified as necessary by an individual or team with "organization-wide responsibility for the SDM" [Henninger, 2001].

### 2.1.4 Summary

After reviewing many of the most significant software processes of various weights, it can be noted that heavy-weight processes tend to focus on the formalities of SPI, but are non-prescriptive in nature and leave the details of lifecycle and activity selection up to an organization to decide. Agile processes rarely offer any insight into the means of tailoring a process, yet require problems or changes to be addressed promptly and continuously throughout a development effort. Some attempts have been made to create custom processes for each project an organization launches. In some cases it was suggested to combine practices from multiple software processes based on a project's needs [Mnkandla, 2008], whereas in others, activities and tasks were prescribed based on whether specific predefined rules applied to the

project at hand [Henninger, 2001].

Table 2.1 summarizes how tailoring is addressed in the software processes discussed in this chapter.

Table 2.1: Tailoring in Heavyweight, Agile, and Custom Built Software Processes

| | **Process/ Framework** | **Does process documentation explicitly address tailoring?** | **When does tailoring occur?** | **What triggers process tailoring?** | **Tailoring level/ granularity** | **Who performs process tailoring?** | **What are the steps taken to tailor the process?** |
|---|---|---|---|---|---|---|---|
| **Heavy weight software processes** | **CMMI [CMMI, 2001]** | Yes | Prior to the beginning of a project or organizational function | The instantiation of a new project | Does not specify which activities to use but rather describes the characteristics that should exist in an organization's set of processes and activities. Tailoring can include modifying a life-cycle model, combining elements of different lifecycle models, modifying process elements. | Does not specify | A defined process is created by selecting the relevant processes and life-cycle elements are from the organization's standard process and tailored to address the needs of a specific project. The defined process is monitored and objectively evaluated to ensure that the deliverables and services it provides meet the required objectives and standards. |
| | **ISO 90003 [ISO/IEC, 2008]** | Yes | When the need arises | Tailoring is a continuous effort that occurs as needed as the project progresses | Does not specify any required criteria for selecting the process or activities to use as long as they are suitable for the nature of the project. | Does not specify | Data is continuously collected and analyzed in order to support effective management of the processes. Tailoring is considered part of quality planning. |
| | **Bootstrap [Kuvaja, 1994]** | Yes | Improvement activities are repeated whenever a problem arises | Does not specify what constitutes a problem in the process. | Offers high-level guidelines to process improvement without specifying the means to achieve them | A specified team | An improvement strategy, improvement model, organizational sponsorship, and an improvement process are required for tailoring. |

| | Process/ Framework | Does process documentation explicitly address tailoring? | When does tailoring occur? | What triggers process tailoring? | Tailoring level/ granularity | Who performs process tailoring? | What are the steps taken to tailor the process? |
|---|---|---|---|---|---|---|---|
| | **SPICE [ISO/IEC, 1998a]** | Yes | When the need arises | To address business goals or other defined needs of an organization | Offers high-level guidelines on continuously monitoring processes and their practices and evaluating them quantitatively to ensure their effectiveness and efficiency in achieving the organization's needs and business goals | A specified team, but tailoring is initiated and supervised by higher management | The organization specifies measurable improvement goals and prioritizes them before an improvement effort is launched. Improvement is considered a project in itself, and the improvement plan must be assessed and analyzed before improvements are prioritized for implementation. Improvement efforts are monitored and compared against the improvement plan to ensure compliance and that the improvement goals are being met. |
| **Agile software processes** | **XP [Dudziak, 1999], [Meso and Jain, 2006]** | No | When the need arises | Not specified | Not specified | Not specified | No steps are mentioned |
| | **Scrum [Abrahamsson, 2002]** | Yes | When the need arises | Daily Scrum Meetings where any deficiencies in the process or practices used are identified and addressed | Not specified | Not specified, but the Scrum Master is responsible for approving any modifications to the process. | No steps are mentioned |
| | **ASD [Highsmith, 2000]** | Yes | At the end of a cycle or at the end of the project | The people-and-process review that occurs at the end of each cycle as well as the end of the project | Not specified | Not specified | Four general questions are suggested as a means of evaluation: what's working?, what's not working?, what do we need to do more of?, and what do we need to do less of? |

| | Process/ Framework | Does process documentation explicitly address tailoring? | When does tailoring occur? | What triggers process tailoring? | Tailoring level/ granularity | Who performs process tailoring? | What are the steps taken to tailor the process? |
|---|---|---|---|---|---|---|---|
| | **FDD [Goyal, 2008], [Abrahamsson, 2002], [Wikipedia, 2015b]** | No | When the need arises | Does not specify | Not specified | Not specified | No steps are mentioned |
| **Automated attempts at process creation and tailoring** | **GAM [Mnkandla, 2008]** | Yes | Before each project begins | The instantiation of a new project | Specifies the practices to be used based on their suitability for a project | A developer or project manager | Classifying projects for the suitability of agile methodologies, determining the agile methodologies appropriate for a project and selecting specific practices from these methodologies to be used, and finally tuning the selected practices |
| | **BORE [Henninger, 2001]** | Yes | Before each project begins and whenever the need arises when a project is underway. | The beginning of a project or if it is determined that the BORE options that were originally selected were incorrect, insufficient or new information surfaced that caused the selected options to require change. | Tailoring can be performed by adding tasks to the SDM and editing the rules for adding tasks to a project | Developers can tailor an project's SDM but escalating these activities to the organization's SDM should be performed by a person or team with organization-wide responsibility for the SDM | A deviation request is generated in order to add new rules and activities to the repository. After the process has been tailored, changes to the process are reviewed carefully to ensure that correct actions have been taken to improve the process before the tailoring is approved. |

Chapter 3

Research Objectives

## 3.1 Research Objectives

The overarching goal of this research was to identify a systematic approach to adjusting software process during the development effort. The specific objectives are:

**Research Objective 1:** Defining a format for cataloging software development practices. This catalog documents key aspects and characteristics of software practices crucial to successfully tailoring a process. Some of these characteristics include the development activity which a given practice can be applied to, the inputs and outputs of the practice, how well the practice achieves a developer's goals, and how it compares to other candidate practices.

**Research Objective 2:** Describing a method for identifying indicators that point to a need to alter the software process. This helps in determining when tailoring is required, and the order in which modifications are introduced to the process, based on the level of pain and other contributing factors.

**Research Objective 3:** Introducing a framework for providing automated help in resolving pain points. The DASP framework was designed to make tailoring recommendations for the developer's current software process. The recommendations are based on attributes and features that the developer values most in a given context or situation. As part of achieving this objective, a methodology for evaluating and selecting an alternate practice from a number of candidate practices is described.

## 3.2 Case Study

We derived the specific research objectives from a case study that aimed to test the concept of tailoring a software process based on pain points in a classroom setting with actual software engineering students. The involvement of software developers in the early stages of developing the framework was important since the DASP framework focuses on value from the developer's perspective. This case study allowed us to gain insight into software developers' willingness to use this tailoring methodology, and gather any feedback that would help better achieve the research objectives and develop a more comprehensive and usable framework.

### 3.2.1 Case Study Objectives

The objective of the case study was to gain insight into the participants' views and attitudes regarding tailoring a software process based on pain points they were experiencing with the software process. The case study was guided by the following objectives:

1. How do students perceive the concept of tailoring a software process based on their pain points?

2. How likely are students to adopt this method of tailoring software process outside the Software Process classroom?

### 3.2.2 Population and Participants

The case study was carried out in the Fall 2015 semester at Auburn University. The population of this case study consisted of 69 students, both graduate and senior year undergraduate, enrolled in two sections of the Software Process course (Comp 5700/6700/6706) offered by the Department of Computer Science and Software Engineering. Students that were enrolled in the course voluntarily offered feedback regarding the course assignments anonymously.

### 3.2.3 Survey

It was determined that the best way to survey students to gain insight into their attitudes regarding tailoring software process based on their pain points, was to collect information voluntarily and anonymously. Since no identifying information would be collected or stored, a Request for Exempt Category Research was submitted to the Auburn University Institutional Review Board (IRB) for Research Involving Human Subjects. The Auburn University IRB approved the protocol entitled "Dynamically Adjustable Software Process"[1], which allowed us to conduct the survey.

The survey administered at the end of the Fall 2015 semester contained eleven questions. Seven of the questions asked students to rank their experiences with different aspects of tailoring the software process on a 5-point Likert scale. The remaining four questions were essay-style questions that aimed to gather more detailed feedback regarding students' pain points and tailoring the software process based on these pain points. The details of the questions in the survey and all possible answers for non-essay questions are presented below:

**The following items assess your impression of the software processed used for CA03 and CA04:**

**Question 1)** What element(s) of the process you used for CA03 caused the most pain? What would you recommend to alleviate the pain?

**Question 2)** What element(s) of the process you used for CA04 caused the most pain? What would you recommend to alleviate the pain?

**Question 3)** In CA04, you had a choice of using either scenarios or user stories to specify requirements. Which alternative did you choose? How well did it support your design (and ultimately your implementation)? Would you use that alternative again if time permitted another assignment? If not, why?

[1] Auburn University IRB Exempt Protocol #16-126 EX 1605

**The following questions assess your impression of process progression across the semester:**

**Question 4)** Adjusting the process from assignment to assignment based on pain points improved my adherence to software process.

*Strongly Agree*

*Agree*

*Neutral*

*Disagree*

*Strongly Disagree*

**Question 5)**Adjusting the process from assignment to assignment based on pain points did not improve my adherence to software process.

*Strongly Agree*

*Agree*

*Neutral*

*Disagree*

*Strongly Disagree*

**Question 6)** Selecting practices to use based on my pain points was in line with my personal style of software development.

*Strongly Agree*

*Agree*

*Neutral*

*Disagree*

*Strongly Disagree*

**Question 7)**Selecting practices to use based on my pain points was not in line with my personal style of software development.

*Strongly Agree*

*Agree*

*Neutral*

*Disagree*

*Strongly Disagree*

**Question 8)**I will likely tailor software processes based on my pain points for future projects outside COMP5700/COMP6700/COMP6706.

*Strongly Agree*

*Agree*

*Neutral*

*Disagree*

*Strongly Disagree*

**Question 9)** I will not likely tailor software process based on my pain points for future projects outside COMP5700/COMP6700/COMP6706.

*Strongly Agree*

*Agree*

*Neutral*

*Disagree*

*Strongly Disagree*

**Question 10)**What you think was the most difficult part of tailoring the software process based on pain points? (Things to consider: frequency of tailoring, assessing pain points, adapting to alternative practices, etc.)

**Question 11)** Based on what you've experienced so far, what is the likelihood that you would voluntarily use the following practices again?

Practices: TDD, Component-Based Estimation, Scenarios, User Stories, Change log, Time log, Historical data, Review checklist, CRC cards.

Likelihood of voluntarily using the practice again:

*Enthusiastic yes!*

*Probably would use it.*

*Could go either way.*

*Probably would not use it.*

*Ick! No way!*

### 3.2.4 Results and Analysis

The first set of questions (Question 1  Question 3) aimed to assess students' impressions of the software processed used for two of their assignments: CA03 and CA04.

**Question 1:** What element(s) of the process you used for CA03 caused the most pain? What would you recommend to alleviate the pain?

**Question 1 Results:**

Figure 3.1 depicts the student responses to which software activity performed for the CA03 assignment they found to be most painful.

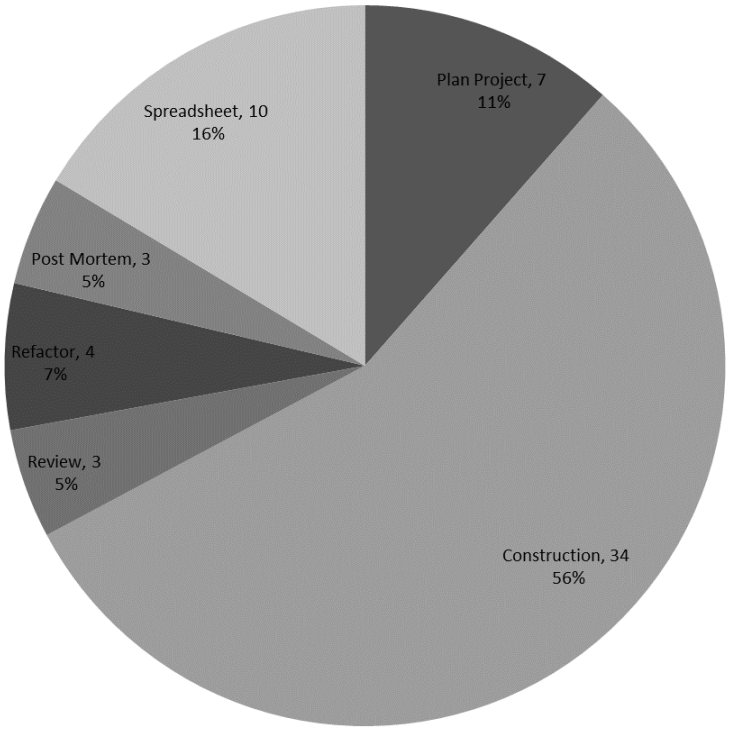


Figure 3.1: Elements of the Process that Caused the Most Pain in CA03

The results from Question 1 show that since a new practice was introduced for the Construction activity in the CA03 assignment, namely TDD, the majority of students found adapting to the new practice to be the most painful aspect, and reported it as their biggest pain point for that iteration. It can also be noted that a number of students found logging their activities in the assigned spreadsheet tedious. As for their recommendations for alleviating the pain, from the 34 students that said the Construction activity was most painful, 10 did not offer any suggestions to alleviate the pain. Of the responses that contained recommendations, 61% suggested to gain more practice and experience in order to alleviate the pain; 22% suggested better analysis; 6% suggested better planning and another 6% suggested changing the TDD practice to another Construction practice. Finally, 5% suggested keeping the process as is and changing nothing. Of the 7 students that reported the Project

Plan as the most painful activity, 67% recommended practicing more, and 33% suggested introducing an Analysis activity.

**Question 2:** What element(s) of the process you used for CA04 caused the most pain? What would you recommend to alleviate the pain?

**Question 2 Results:**

For the CA04 assignment, a new activity, Analysis, was introduced. For this newly introduced activity, students were given the choice to select one of two practices: scenarios or user stories. As for the previous assignment, students were asked to report on the most painful aspect of the CA04 assignment. The results from Question 2 are displayed in Figure 3.2.

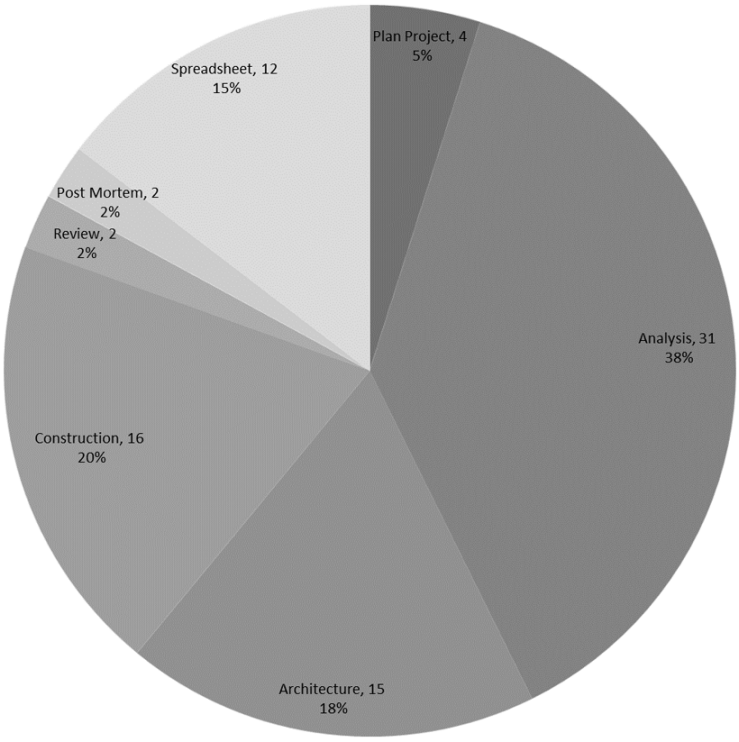


Figure 3.2: Elements of the Process that Caused the Most Pain in CA04

Results from Question 2 show that 38% of students found the newly introduced software activity, Analysis, to be the most painful; 20% of students continued to report Construction as the most painful activity; 15% reported the documentation and logging of activities as the

most painful aspect, a 1% decrease from the CA03 assignment. Planning was found to be the most painful part of the process by 5% of the students; the review activity was reported by 2% as their most painful activity, and another 2% reported the post mortem activity as their most painful activity. When asked what they would recommend to alleviate the pain, only 45% of students offered a response. Of the 31 students that reported analysis, the newly introduced activity, as the most painful activity for this iteration, 29% had no recommendations to alleviate the pain, which shows that software developers can often identify pain points but do not know how to alleviate the pain. Of the students that reported analysis as the most painful activity and offered recommendations, 41% suggested more practice would alleviate the pain. It can be observed that many of the students realize that newly introduced activities may cause discomfort that requires tolerating the pain to become more proficient with the activity.

**Question 3:** In CA04, you had a choice of using either scenarios or user stories to specify requirements. Which alternative did you choose? How well did it support your design (and ultimately your implementation)? Would you use that alternative again if time permitted another assignment? If not, why?

**Question 3 Results:**

For the CA04 assignment, where the analysis activity was introduced to the process, students were allowed to select a practice from two candidate practices to carry out the analysis activity: scenarios or user stories. User stories were selected by 62% of the students, whereas only 20% used scenarios. In addition, 15% of the students elected to use both scenarios and user stories since they did not know which of the two would serve their needs better. Finally, 2% of students deviated from the process and did not use either of the two practices for analysis. Figure 3.3 shows the percentages of choices students made to execute the analysis activity.

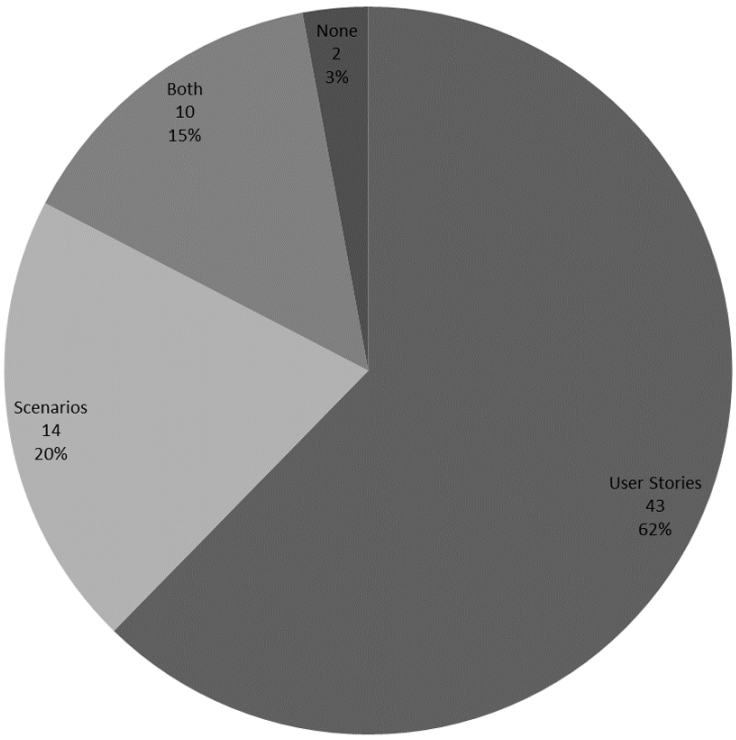


Figure 3.3: The Alternatives Students Chose for Analysis in CA04

Of the 43 Students that used user stories for the CA04 assignment, 81% said they would use user stories again; 7% said that they would use scenarios in the future, and 12% did not specify what they would use for the analysis activity in future projects. Of the 14 students that used scenarios, 43% said they would use scenarios for future projects, 28.5% said they would use user stories, and 28.5% did not specify what they would use. Of the 10 students who used both user stories and scenarios, 30% said they would use both again; 20% said they would use user stories, while 20% said they would use scenarios. Finally, 30% did not specify what they would use for future projects.

The second set of questions (Question 4  Question 9) aimed to assess students' attitudes regarding the concept of tailoring software process based on pain points. Each two questions presented to the students aimed to elicit the same information by rephrasing the question. This allowed us to ensure the integrity of the data being gathered, and disregard

any incoherent or contradictory responses where students gave two negative or two positive answers to questions that should have opposite answers. The results presented show student responses after disregarding the incoherent data.

**Question 4:** Adjusting the process from assignment to assignment based on pain points improved my adherence to software process.

**Question 4 Results:**

From the results displayed in Figure 3.4 it can be noted that 70% of the students' responses were positive when asked whether having the ability to adjust the software processed based on their pain points improved their adherence to software process. Neutral responses were reported by 21% of the students, and only 9% did not think that this method of adjusting the software process improved their adherence to it.

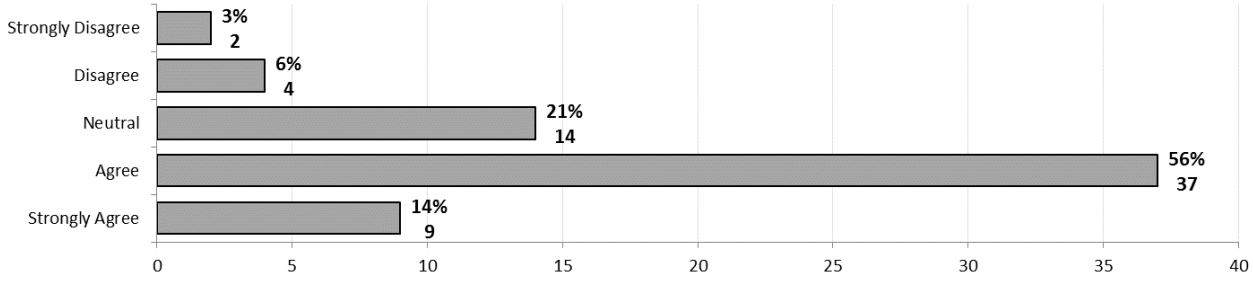


Figure 3.4: Adjusting the Process Based on Pain Points Improved My Adherence to Software Process

**Question 5:** Adjusting the process from assignment to assignment based on pain points did not improve my adherence to software process.

**Question 5 Results:**

Question 5 aimed to elicit the same information from students as Question 4 by rephrasing the question. It can be seen from Figure 3.5 that similar responses were recorded with 68% of the responses being positive, 21% being neutral, and 11% being negative.

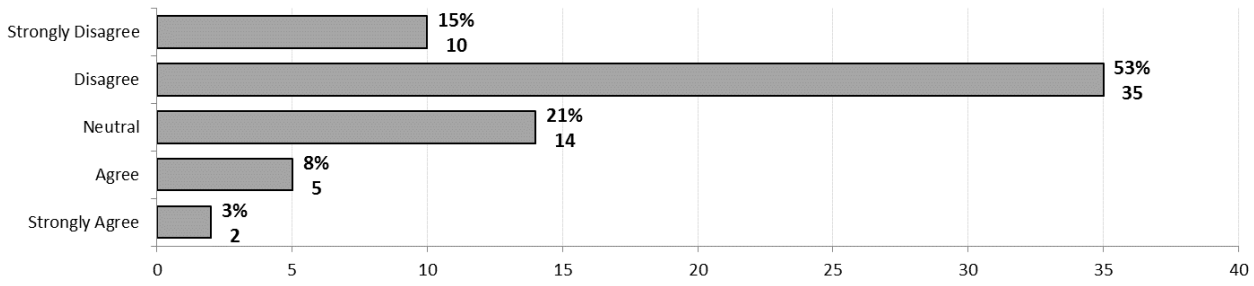


Figure 3.5: Adjusting the Process Based on Pain Points DID NOT Improve My Adherence to Software Process

After analyzing the results from Questions 4 and 5 and taking the averages of each of the responses, Figure 3.6 shows the students' responses as to whether adjusting a software process based on pain points improved their adherence to the software process. An average of 69% of students' responses agreed or strongly agreed with this statement; 21% were neutral, and only 10% of students disagreed or strongly disagreed with this statement.

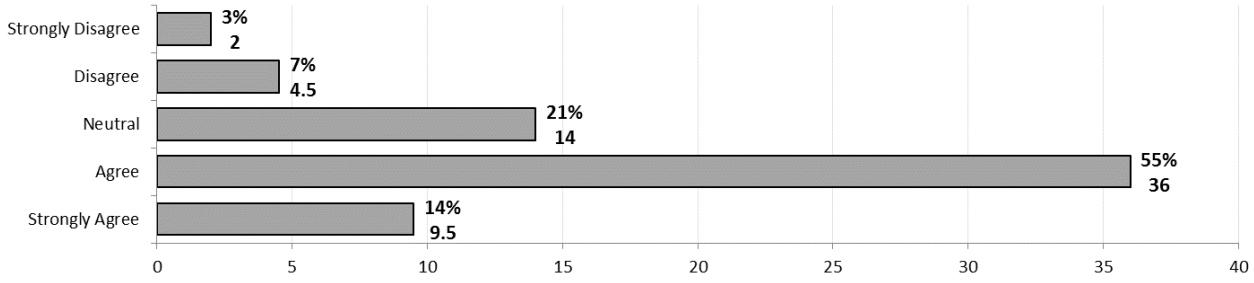


Figure 3.6: Average Opinions on: Adjusting the Process Based on Pain Points Improved My Adherence to Software Process

**Question 6:** Selecting practices to use based on my pain points was in line with my personal style of software development.

**Question 6 Results:**

When asked whether making process adjustments based on their pain points was in line with their personal style of software development, 67% of students had positive responses and

agreed with this statement; 21% had neutral responses, whereas only 11% of the responses were negative. Figure 3.7 details student responses to Question 6.

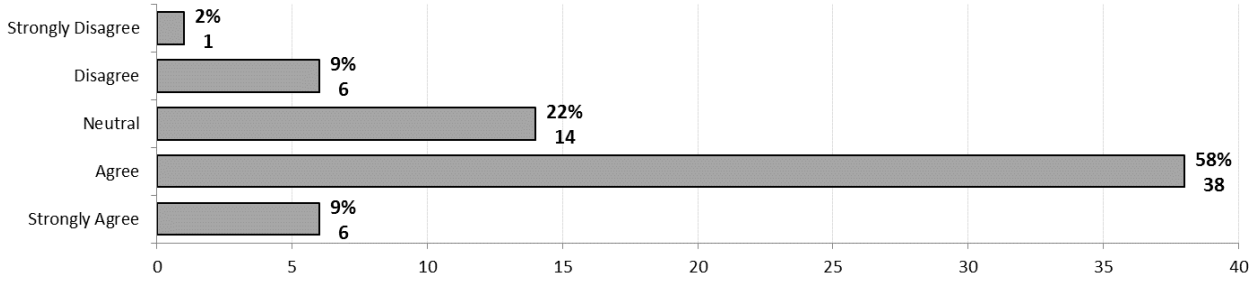


Figure 3.7: Selecting Practices Based on Pain Points was in Line with My Personal Style of Software Process

**Question 7:** Selecting practices to use based on my pain points was not in line with my personal style of software development.

**Question 7 Results:**

Question 7 aimed to elicit the same information from students as Question 6 by rephrasing the question. It can be seen from Figure 3.8 that similar responses were recorded with 57% of the responses being positive, an increase to 34% in neutral responses, and the remaining 10% of the responses being negative.

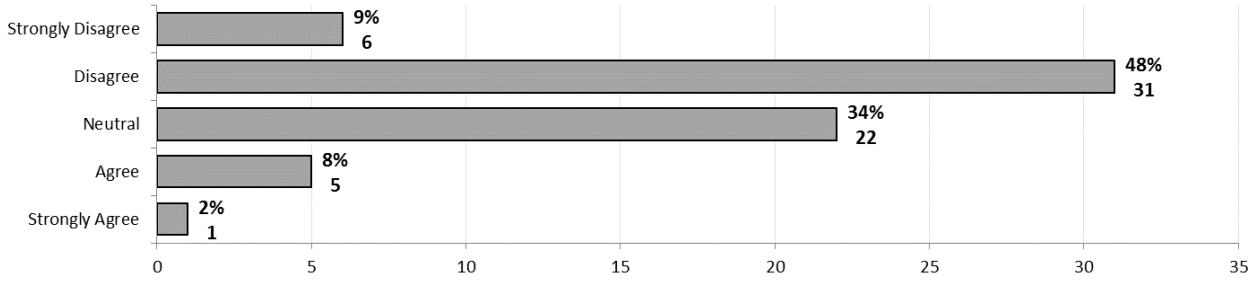


Figure 3.8: Selecting Practices Based on Pain Points was NOT in Line with My Personal Style of Software Process

After analyzing the results from Questions 6 and 7 and taking the averages of each of the responses, Figure 3.9 shows the students' responses as to whether adjusting a software

process based on pain points was in line with their personal styles of software development. It can be observed that an average of 62% of students either agreed or strongly agreed with this statement; 28% were neutral, and only 10% of the students either disagreed or strongly disagreed with this statement.

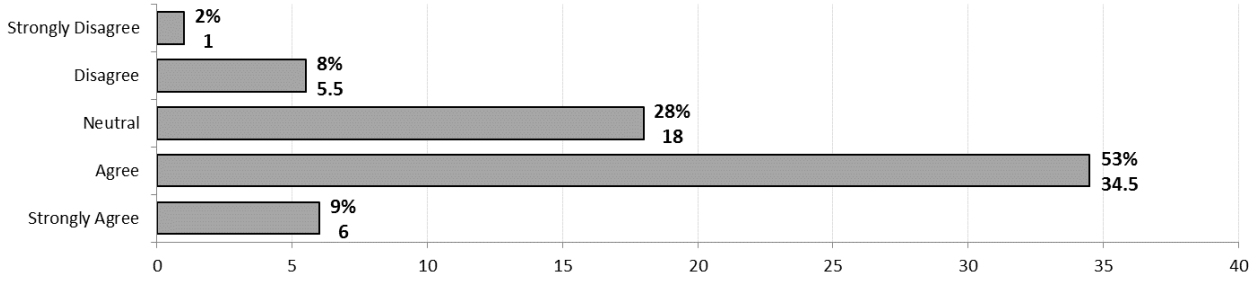


Figure 3.9: Average Opinions on: Selecting Practices Based on Pain Points was in Line with My Personal Style of Software Process

**Question 8:** I will likely tailor software processes based on my pain points for future projects outside COMP5700/COMP6700/COMP6706.

**Question 8 Results:**

Question 8 aimed to find out the likelihood of students using the concept of tailoring their software process based on their pain points in the future outside of the Software Process (COMP5700/COMP6700/COMP6706) course. The responses were extremely encouraging as can be seen in Figure 3.10 with 88% of the responses being positive stating that they would likely use this method of tailoring in the future; 9% of the responses were neutral, and only 3% of the students did not anticipate using this technique in the future.

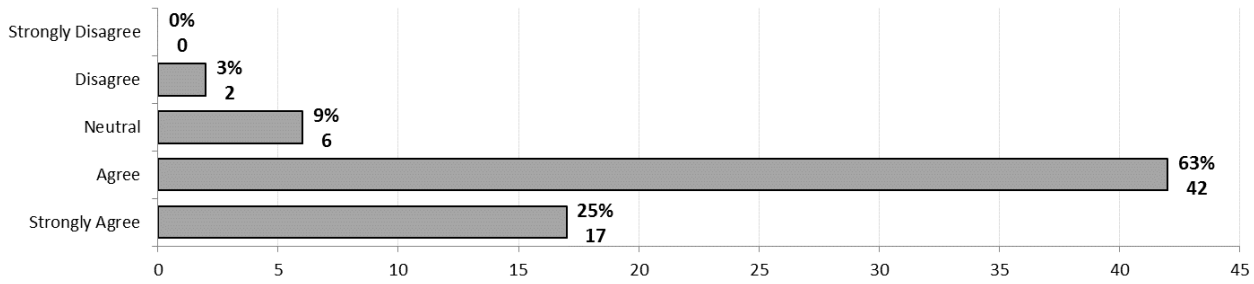


Figure 3.10: I Will Likely Tailor Software Process Based on My Pain Points in the Future

**Question 9:** I will not likely tailor software process based on my pain points for future projects outside COMP5700/COMP6700/COMP6706.

**Question 9 Results:**

Question 9 aimed to elicit the same information from students as Question 8 by rephrasing the question. It can be seen from Figure 3.11 that similar responses were recorded with 76% of the responses being positive, an increase to 19% in neutral responses, and only 4% of the responses were negative.

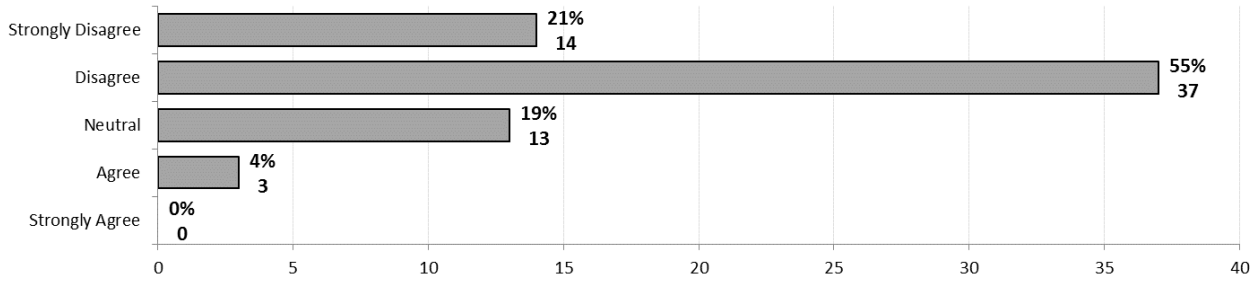


Figure 3.11: I Will NOT Likely Tailor Software Process Based on My Pain Points in the Future

After analyzing the results from Questions 8 and 9 and taking the averages of each of the responses, Figure 3.12 depicts the students' responses as to whether they are likely to tailor software process based on their pain points outside the Software Process course (COMP5700/COMP6700/COMP6706). An average of 82% of students either agreed or

strongly agreed with this statement; 14% were neutral, and only 4% of the students disagreed, none having strongly disagreed with this statement.

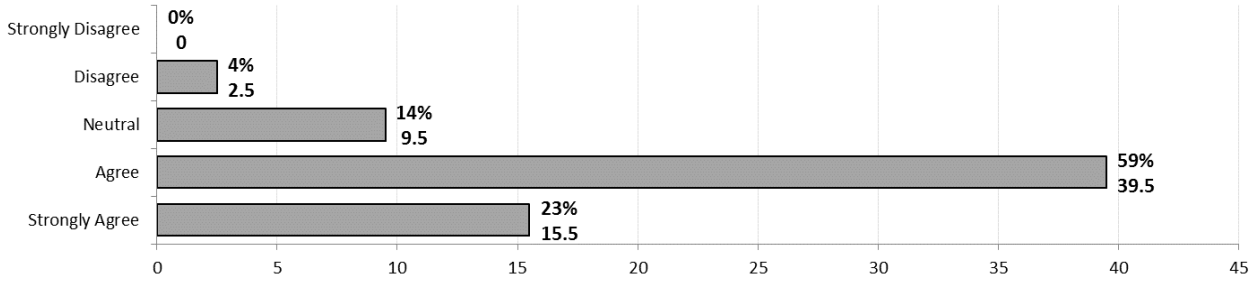


Figure 3.12: Average Opinions on: I Will Likely Tailor Software Process Based on My Pain Points in the Future

**Question 10:** What you think was the most difficult part of tailoring the software process based on pain points? (Things to consider: frequency of tailoring, assessing pain points, adapting to alternative practices, etc.)

**Question 10 Results:**

Question 10 was an essay-style question with the objective of finding out what students found most difficult about tailoring a software process based on pain points. Examples of difficulties were given I the questions but students could also give their own. Figure 3.13 shows the difficulties students reported, and the percentage of students that reported each. Students were allowed to report more than one difficulty. It can be observed that students found assessing pain points to be the most painful aspect, followed by adapting to newly introduced practices and the frequency in which tailoring occurred respectively.

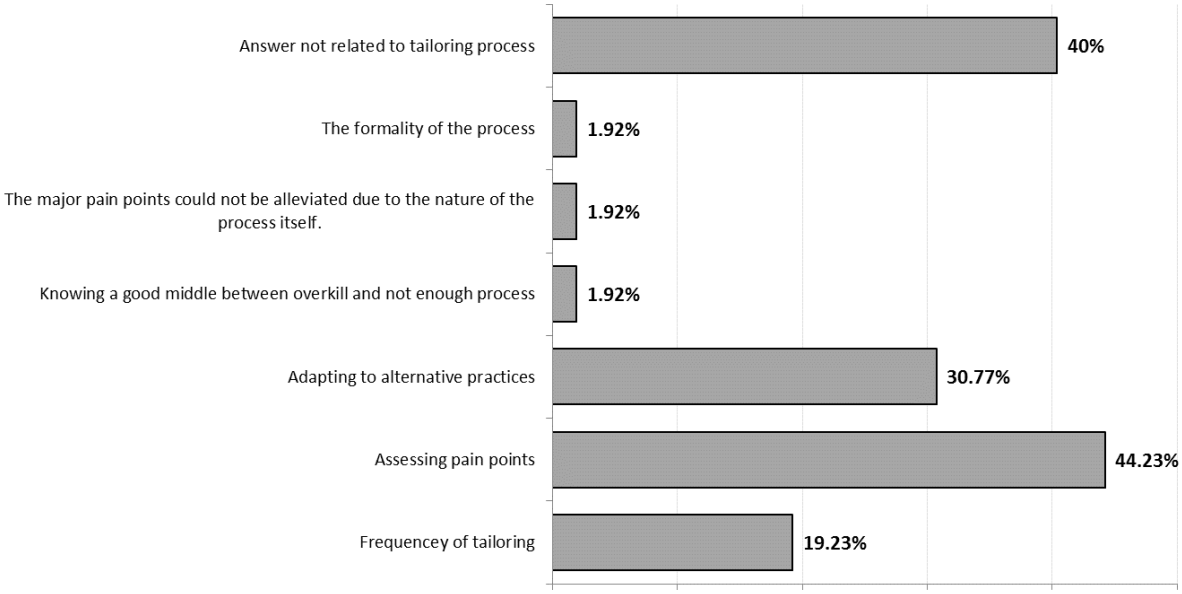


Figure 3.13: What Students Perceived as the Most Difficult Part of Tailoring Process Based on Pain Points

**Question 11:** Based on what you've experienced so far, what is the likelihood that you would voluntarily use the following practices again? TDD, Component-Based Estimation, Scenarios, User Stories, Change log, Time log, Historical data, Review checklist, CRC cards.

**Question 11 Results:**

**TDD:**

Figure 3.14 shows the extremely enthusiastic responses regarding using TDD voluntarily in the future, as 100% of students said that they would probably use it, despite it being reported as a pain point throughout the semester.

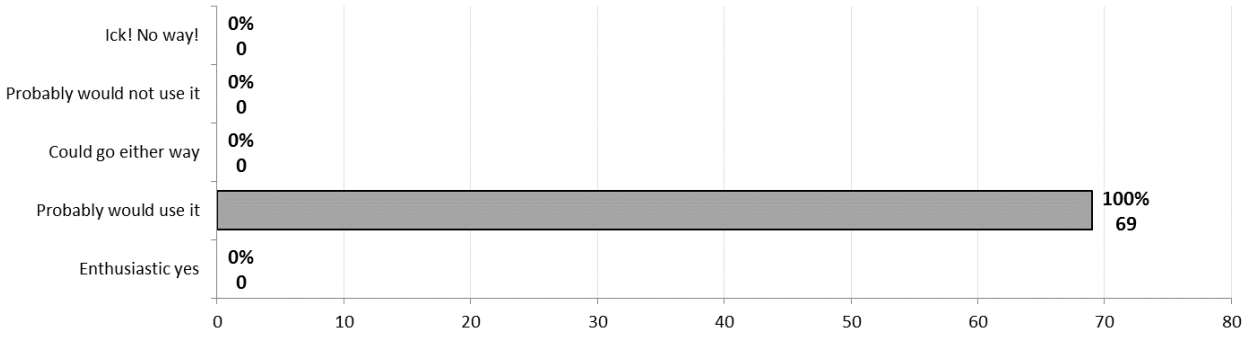


Figure 3.14: Students' Likelihood of Using TDD Outside the Software Process Class

**Component-Based Estimation:**

Figure 3.15 shows student responses regarding their likelihood of using Component-Based Estimation. It can be seen that 13% were extremely enthusiastic about it; 43% said they would probably use it; 28% were undecided as to whether they would use it voluntarily in the future; 12% of students said they probably would not use it, and 4% said they definitely would not. Perhaps the less enthusiastic responses came due to the fact that Component-Based Estimation required a lot of logging in the assignments' Excel spreadsheets which many students reported as a pain point.

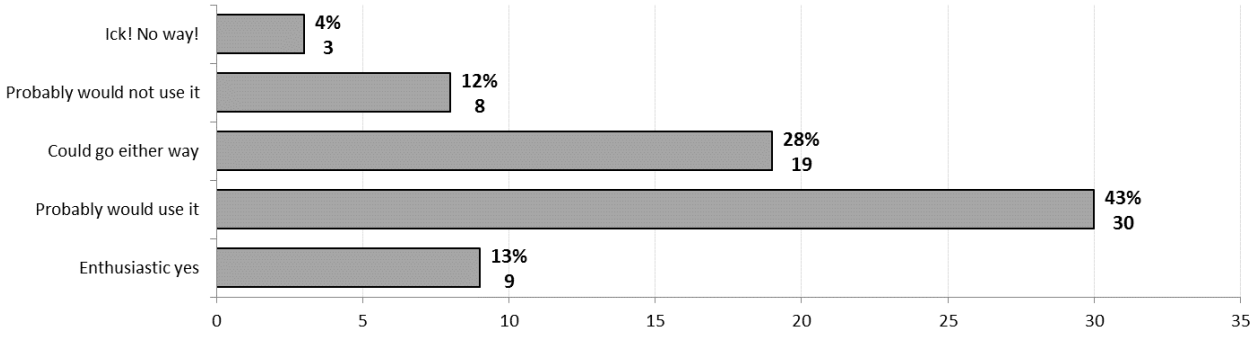


Figure 3.15: Students' Likelihood of Using Component-Based Estimation Outside the Software Process Class

**Scenarios:**

When asked about the use of Scenarios, it can be seen in Figure 3.16 that students had mixed responses and no specific trend could be observed. It can be observed that 13% were extremely enthusiastic about Scenarios; 33% said they would probably use them; 22% were undecided as to whether they would use them voluntarily in the future; 25% of students said they probably would not use Scenarios, and 7% said they definitely would not.

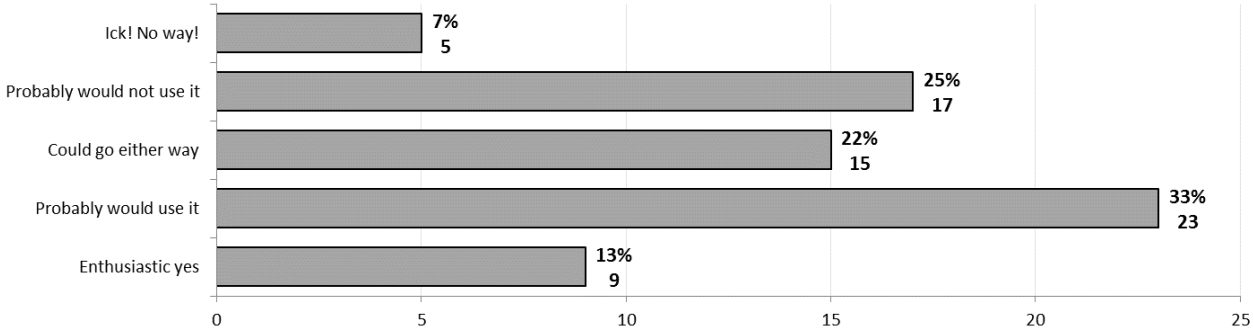


Figure 3.16: Students' Likelihood of Using Scenarios Outside the Software Process Class

**User Stories:**

Students seemed more enthusiastic about User Stories than they were about Scenarios, as 18% were extremely enthusiastic about User Stories; 45% said they would probably use them; 16% were undecided as to whether they would use them voluntarily in the future; 10% of students said they probably would not use User Stories, and 3% said they definitely would not. Figure 3.17 depicts students' attitudes about User Stories.

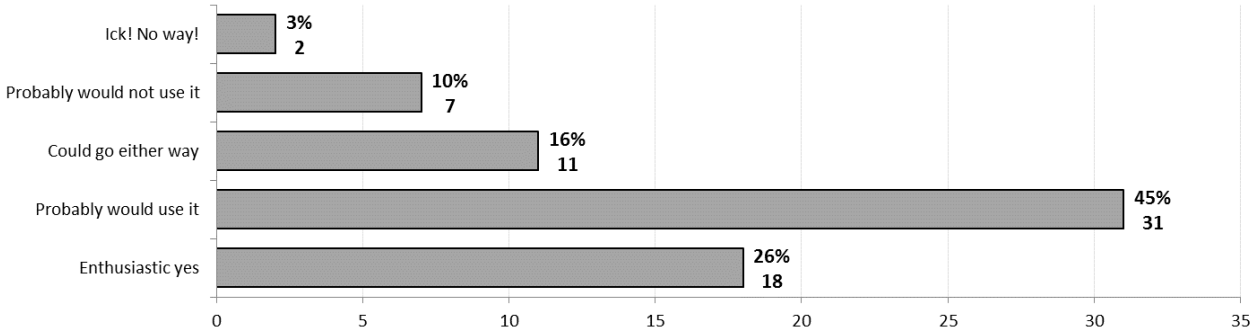


Figure 3.17: Students' Likelihood of Using User Stories Outside the Software Process Class

**Change Log:**

Figure 3.18 shows students' attitudes with regard to the Change Log practice. It can be seen that 10% were extremely enthusiastic about using a Change Log in the future; 36% said they would probably use one; 25% of the students were undecided as to whether they would a Change Log; 10% of students said they probably would not use one, and another 10% said they definitely would not.

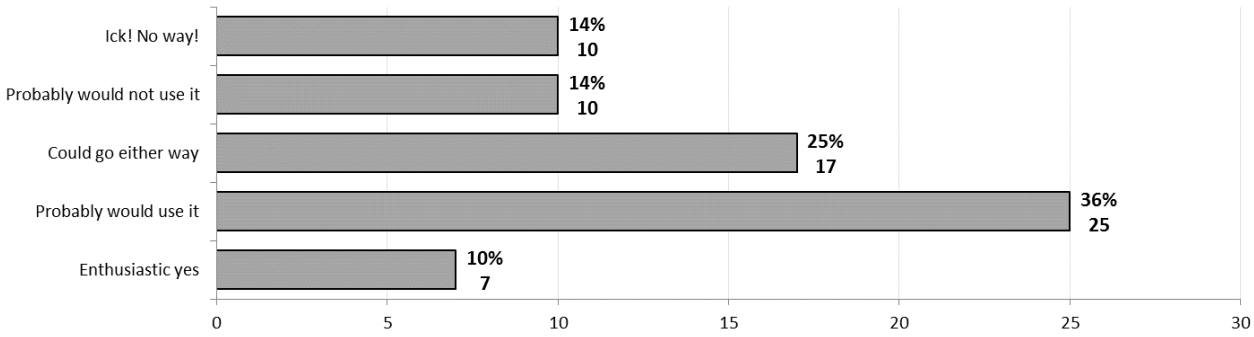


Figure 3.18: Students' Likelihood of Using Change Logs Outside the Software Process Class

**Time Log:**

When asked about using a Time Log for future projects, 25% were extremely enthusiastic about using one; 28% said they would probably use one; 22% of the students were undecided as to whether they would a Time Log voluntarily; 14% of students said they probably would not use a Time Log, and 10% said they definitely would not. Figure 3.19 below displays these results.

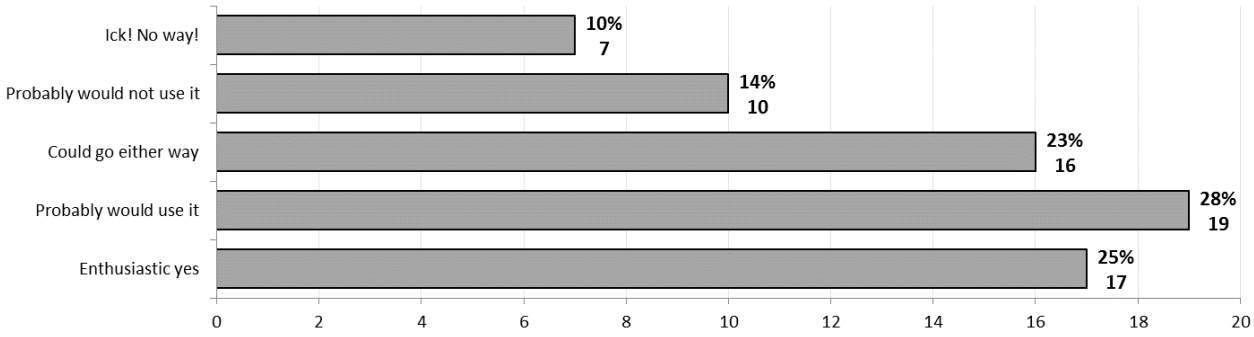


Figure 3.19: Students' Likelihood of Using Time Logs Outside the Software Process Class

**Historical Data:**

From Figure 3.20 it can be observed that many students saw benefit in using Historical Data for future projects, as 28% of them said they would definitely use it; 36% of students said they would probably use Historical Data; 25% were unsure of whether or not they would use this practice; 9% said they would probably not use Historical Data, and 3% said they would definitely not.

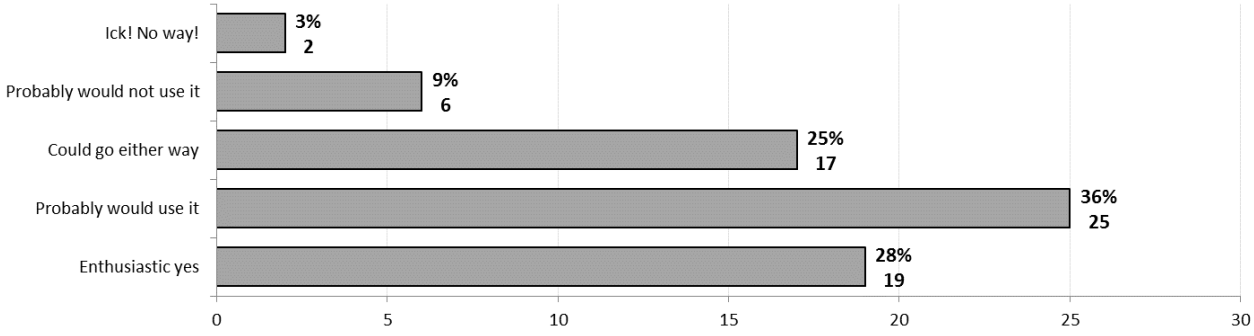


Figure 3.20: Students' Likelihood of Using Historical Data Outside the Software Process Class

**Review Checklist:**

Students' attitudes pertaining to the use of Review Checklists in future projects can be seen in Figure 3.21. The majority of students had positive responses as 20% of them said

they would definitely use Review Checklists; 38% said they would probably use them; 19% were undecided as to whether they would use this practice; 22% of students said they would probably not use Review Checklists, and only 1% said they would definitely not.

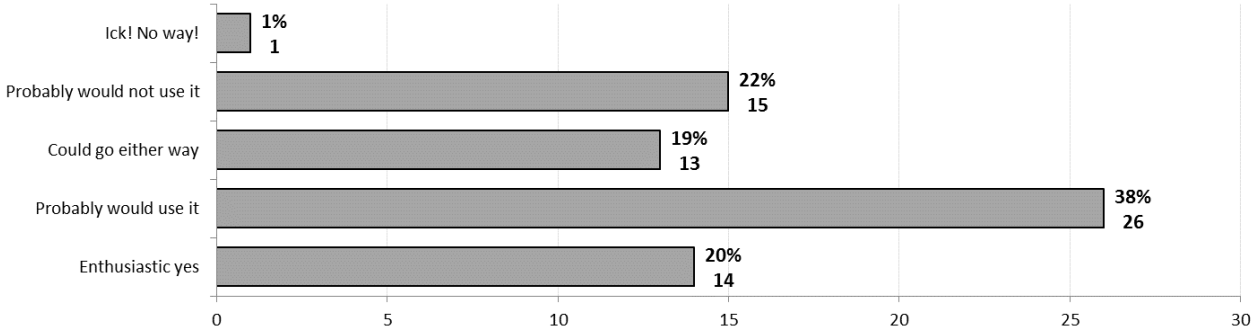


Figure 3.21: Students' Likelihood of Using Review Checklists Outside the Software Process Class

**CRC Cards:**

Figure 3.22 shows a majority of positive responses with reference to CRC Cards with 25% of students saying that they would definitely use CRC Cards again; 30% said that they would probably use them; 26% of students were undecided about the use of CRC Cards; 14% said that they would probably not use them in the future, and 4% said they would definitely not consider using them.

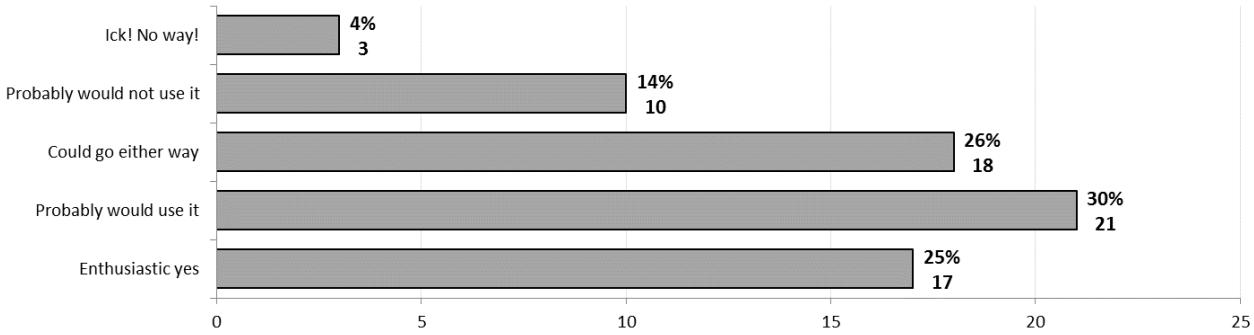


Figure 3.22: Students' Likelihood of Using CRC Cards Outside the Software Process Class

### 3.2.5 Case Study Conclusions

This case study shed light onto the value of tailoring a software process based on a software developer's pain points. It successfully achieved its objectives of determining students' attitudes regarding the concept of tailoring a software process based on their pain points, and discovering the likelihood of students adopting this method of tailoring software process outside the Software Process classroom.

By asking software developers what they found to be the most painful aspects of the software process after each iteration, it was found that the introduction of new practices is usually reported as a pain point, but many software developers acknowledge that more practice using those software practices will alleviate the pain. This was found to be the case when introducing the TDD practice in the CA03 assignment, and the User Stories/Scenarios practices in the CA04 assignments. In both cases, the software activity containing the newly introduced practice was reported as the most painful aspect of the process for that iteration.

When asked about what they considered to be the most difficult part of tailoring software process based on pain points, students reported assessing pain points to be the most painful aspect, followed by adapting to newly introduced practices and the frequency in which tailoring occurred respectively. This demonstrates that assistance with determining the source of pain would be a valuable feature of the DASP framework. It is important to note that the introduction of a new practice into a software process hardly occurs without some degree of pain, which will usually gradually subside as the developer becomes more accustomed to the use of the practice. Most practices take multiple iterations for the software developer to adapt to them. This fact was taken into consideration when designing the DASP framework.

Based on students' attitudes toward the various practices they used throughout the semester, it can be observed that despite students reporting specific practices as pain points, throughout multiple iterations in some cases, many of them saw value in continuing to use these practices outside the Software Process course. For this reason, designing a pain scale

for use by software developers when reporting a pain point is an important contribution that will help indicate whether the developers consider the pain to be temporarily tolerable or require the process to be tailored immediately.

The case study described in this chapter yielded affirmative results that proved the validity and importance of developing the DASP framework. The majority of participating students had positive attitudes regarding tailoring a software process based on their pain points, and most were willing to follow this tailoring strategy outside of the Software Process course (COMP5700/COMP6700/COMP6706). These positive results provided us with more confidence in our research objectives and confirmed our enthusiasm about building a framework to assist software developers in modifying their software processes based on their pain points, and aid them in finding candidate practices that best fit their needs, skills, and experiences.

# Chapter 4

# The DASP Framework

## 4.1 Introduction

The state-of-the-practice in tailoring software processes consists of two schools of thought: one advocates software process adjustment at the end of a software project or iteration, and the other advocates process adjustment in an ad hoc fashion or at set intervals of time often spanning multiple projects. This research proposes a middle ground, in which process adjustment is systematic, responsive, and offers solutions based on evidence.

## 4.2 Context and Rationale: PCSE

Practitioner Centered Software Engineering (PCSE) is a personal software process developed at Auburn University, and has evolved over the years into a process framework that can be tailored to meet a developer's particular needs. PCSE offers a lightweight, less invasive alternative to PSP, and manages to incorporate a number of core CMMI elements while remaining agile. Its unique balance of structure and agility, in addition to its tailoring flexibility makes PCSE the perfect candidate for dynamically tailoring software process.

The core tenet of PCSE software development is that four engineering activities are always carried out in any software development effort regardless of how the process is defined: envision, synthesize, articulate, and interpret. The four fundamental activities involve requirement analysis, a project design or plan, the actual execution of the project, and some form of verification and validation. The nature of these activities and the sequence in which they are carried out varies depending on the development effort itself, the individuals carrying it out, the customer, etc. Each activity is carried out in a certain way, referred to as

a "practice", and it is possible that an activity can be accomplished by a variety of different practices. The mentioned activities, practices, and sequence can be tailored based on pain points and quality goals which are identified, measured, and adjusted throughout the project.

PCSE is comprised of four essential elements (Figure 4.1):

- Minimal Guiding Indicators (MGIs) that keep track of the project and product goals. MGIs guarantee that quality goals are being met, and are measured at various points throughout the process for possible pain points.

- Minimally Sufficient Activities (MSAs) are the minimal subset of activities that are needed in order to carry out the development effort while achieving the required MGI metrics being monitored.

- Minimally Viable Process (MVP) defines the relationship between the MSAs and the sequence in which the MSAs will be performed.

- Minimally Effective Practices (MEPs) are associated with each of the MSAs and describe the manner in which an MSA will be achieved.

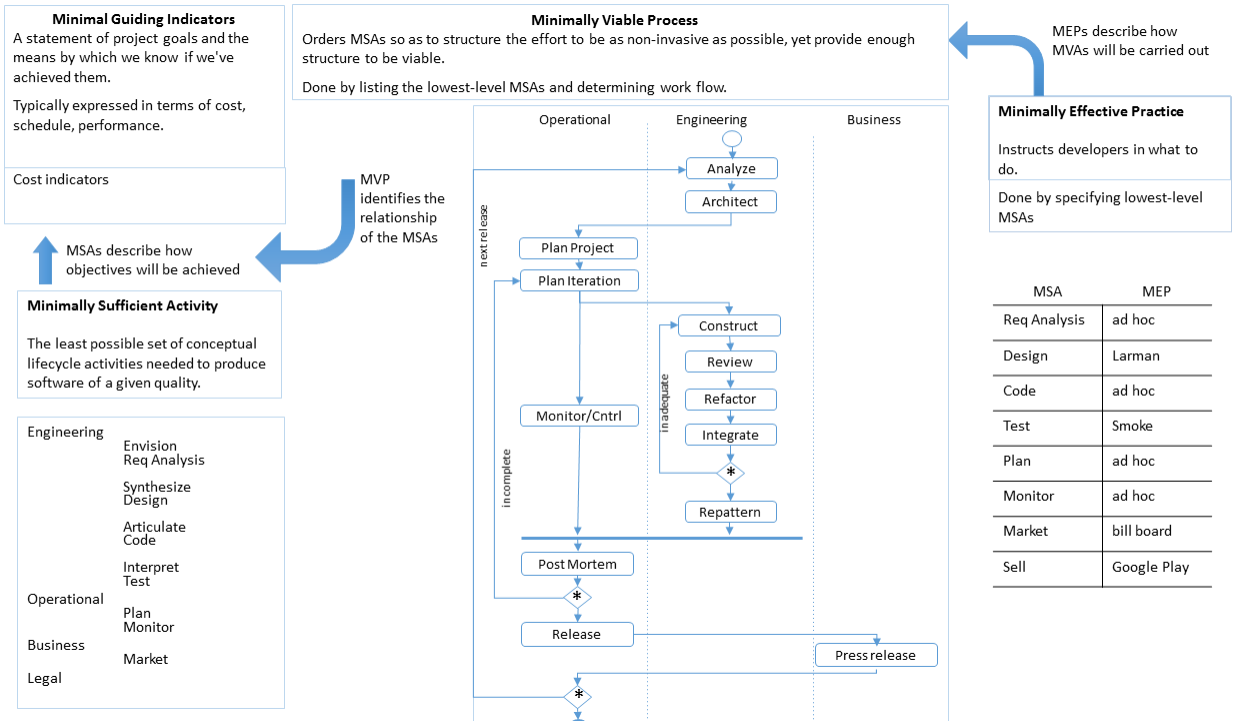


Figure 4.1: The Essential Elements of PCSE

## 4.3 Dynamically Adjustable Personal Software Process

### 4.3.1 The Big Picture

Due to the great variation in software developers' experiences, backgrounds, habits, and abilities to adapt to change, in addition to the wide range of tools available, development environments used, and software project types, there is no silver bullet when it comes to software process. Each developer has preferences, and each project has a unique set of requirements.

Dynamically adjustable personal software process is a novel concept that adds a new dimension to personal software process. It allows for systematic and prompt adjustment of process during the course of a project. This is a fresh approach to software process tailoring, as tailoring typically occurs after the completion of a development effort or at ad hoc intervals that could span multiple projects in some cases. This introduced method of

software process tailoring is iterative, allowing for the process to be adjusted in a dynamic and adaptive manner that caters to a developer's preferences and pain points. The process is continuously monitored for indicators (MGIs) that tailoring is needed, and is adjusted accordingly. Modifications to the process can include anything from the replacement of a specific practice (MEP) with one the developer is more comfortable with, to introducing new activities (MSAs) to the current instance of the process. Figure 4.2 demonstrates the continuous tailoring process of Dynamic PCSE and the elements it encompasses.

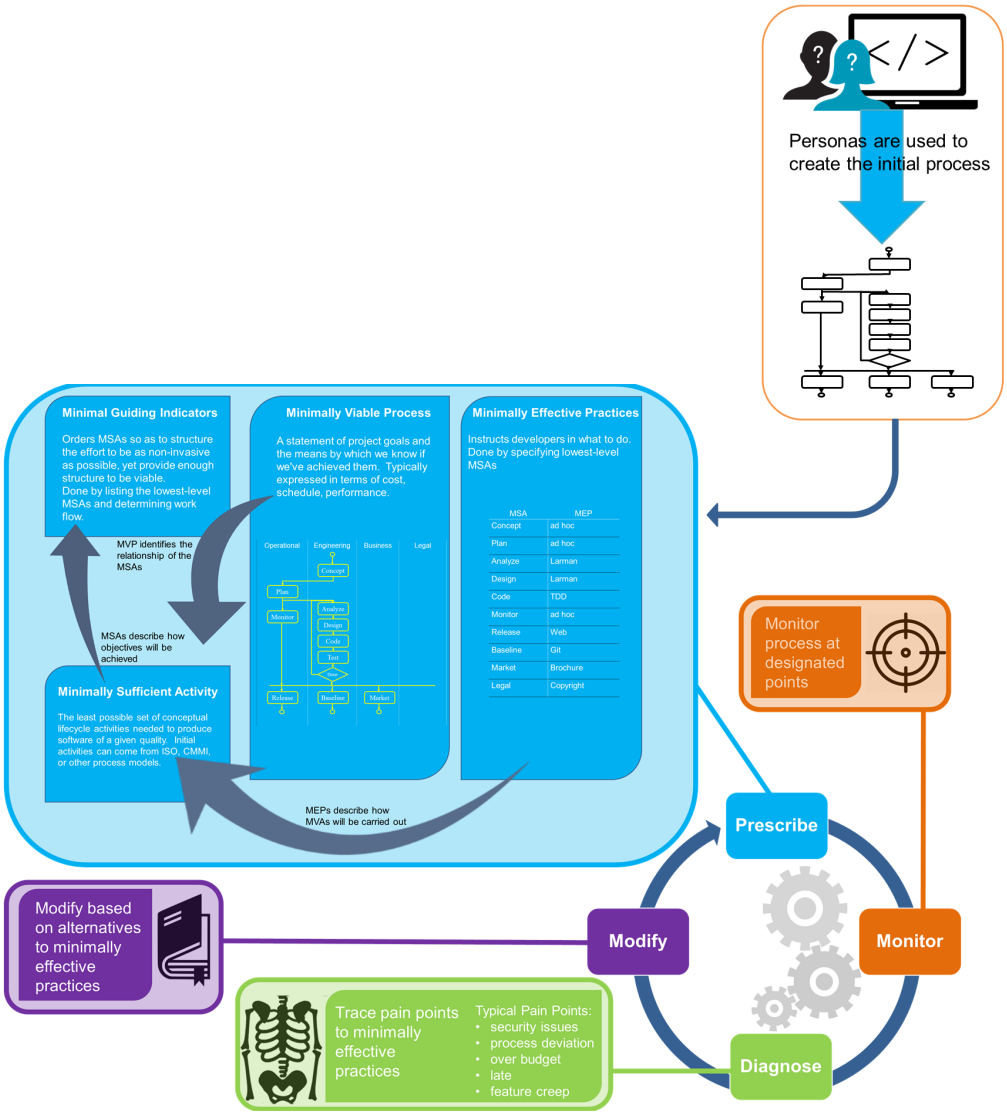


Figure 4.2: Elements of the DASP Framework

Dynamically Adjustable Software Process (DASP) suggests an initial software process to a software developer by matching the software developer to a persona. Ideally, there would exist a pool of personas, a repository of prebuilt software developer profiles that were created by surveying large numbers of software developers from diverse backgrounds and levels of experience. For each persona there would exist a recommended PCSE MVP with a minimal set of MSAs and MEPs. This MVP would serve as a starting point for creating a suitable software process that best reflects a developer's experience and project requirements. Before prescribing the initial software process, the developer answers a number of survey questions to provide input about the recommended MEPs. If the developer is more comfortable with alternate practices, the process can undergo initial tailoring before it is used. The developer now has an MVP with a set of MSAs and MEPs to use as his/her initial personal process as the project begins.

The efficiency and effectiveness of the recommended software process are monitored throughout the course of the project. A set of key metrics representing quality goals and pain points are continuously monitored for the evaluation of the process. This ensures that the process in place is positively contributing to the progress and quality of the software being developed. The developer's conformance to the process is also monitored for any indicators that the activities or practices currently in place need to be revised. A deviation from the process represents a pain point that needs to be addressed. When the need for tailoring arises, the software process is modified accordingly. If multiple adjustments need to be made to the process, they are prioritized based on the level of pain they are causing, reflected in the MGIs. Changes to the process can be introduced gradually to reduce the cost of adapting to the modified process and alleviate the amount of pain and frustration experienced by the developer. The MVP can be modified by adding MSAs and/or replacing some of the current MEPs with alternate MEPs. This iterative cycle of continuously monitoring MGIs and the developer's interaction with the process, and prescribing adjustments that fit the developer's

specific needs and are compatible with the existing MEPs in the process is the main idea behind the DASP approach.

## 4.4 DASP - A Dynamic Framework for Tailoring Software Process

There are many dimensions to making adjustments to a software process. Following the concept of process minimalism means that at early stages of a project, a software developer may be performing many of the four core engineering activities (envision, synthesize, articulate, and interpret) in an ad hoc fashion. As the need arises and pain points begin to surface, modifications to the process may be required in the form of introducing more structured practices and additional guiding indicators. The Dynamically Adjustable Software Process (DASP) framework was created to aid software developers in tailoring their software processes to better fit their needs and software development styles. It allows software developers to make informed software practice selections by comparing the fitness of the different candidate software practices for their process. The comparison is based on a set of criteria derived from the software developers' goals and priorities.

The DASP framework consists of three major components: a catalog of practices, a pain scale for identifying indications that point to the need for tailoring, and a framework for evaluating candidate practices and making a recommendation.

### 4.4.1 Representation of Practices as Catalog Entries

Software practices are associated with software development activities and describe the method in which a software activity is carried out. For each activity, multiple practices can be used, each with different requirements and constraints. The developer often needs to select the most suitable practice from several candidate practices. In order to provide the developer with insight into the suitability of a specific practice for their current process and its compatibility with their existing practices and historical data, we have designed a format for cataloging practices. The purpose of the catalog is to preserve any practice information

relevant to the tailoring process, making it readily available and simplifying the selection of a practice. Each practice is represented as a catalog entry in the practice catalog.

The Entry-Task-Validation-Exit (ETVX) model [Radice et al., 1985] was chosen to represent the practices available for each activity. ETVX highlights the important information that a developer needs to consider when making a practice selection. For each practice, there are activity prerequisites that must be satisfied, such as specific data that may be required in a specific format, in order to enable a developer to complete the tasks of a practice successfully. The details of how a practice is carried out and how to validate that the practice has been carried out successfully are also described in the ETVX model. Figure 4.3 shows the ETVX paradigm and "indicates the relationships and flow among the four aspects of an activity" [Radice et al., 1985].

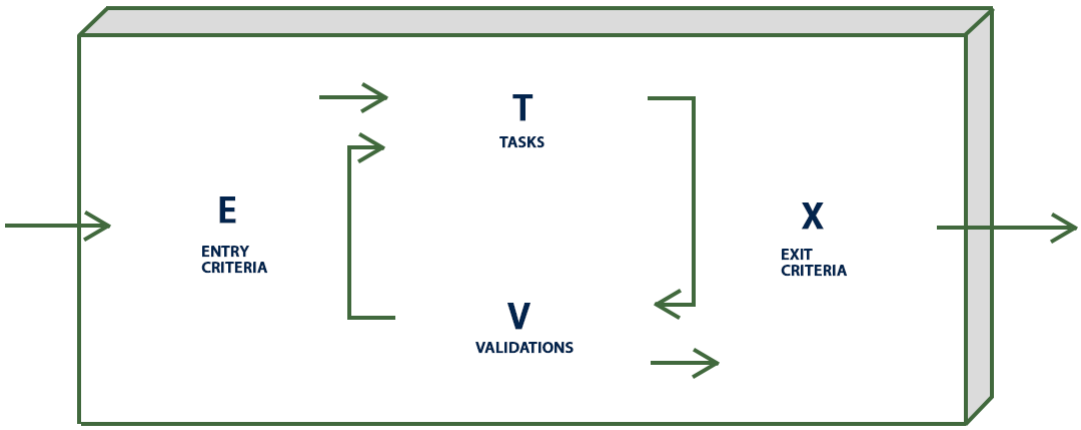


Figure 4.3: The ETVX Paradigm [Radice et al., 1985]

The goal of the practice catalog is to provide developers with insight into the data required to successfully use a practice, what to expect as the outputs of the practice, and any loss of existing data that may occur when transitioning to that specific practice. This will aid developers in deciding whether the practice is a good match for their needs. Developers

can make informed decisions when tailoring their process and can assess whether a practice is in line with their goals and current ways of working.

Due to the dynamic nature of the DASP framework, and the large variation in the suitability of a practice depending on a developer's experience and process goals, each catalog entry contains both static and dynamic sections. The static sections contain invariable information that applies to a practice at all times, whereas the content of the dynamic sections will vary based on a developer's goals, pain points, current practices, and project requirements. Figure 4.4 shows the layout of a catalog entry and points out the static and dynamic parts.

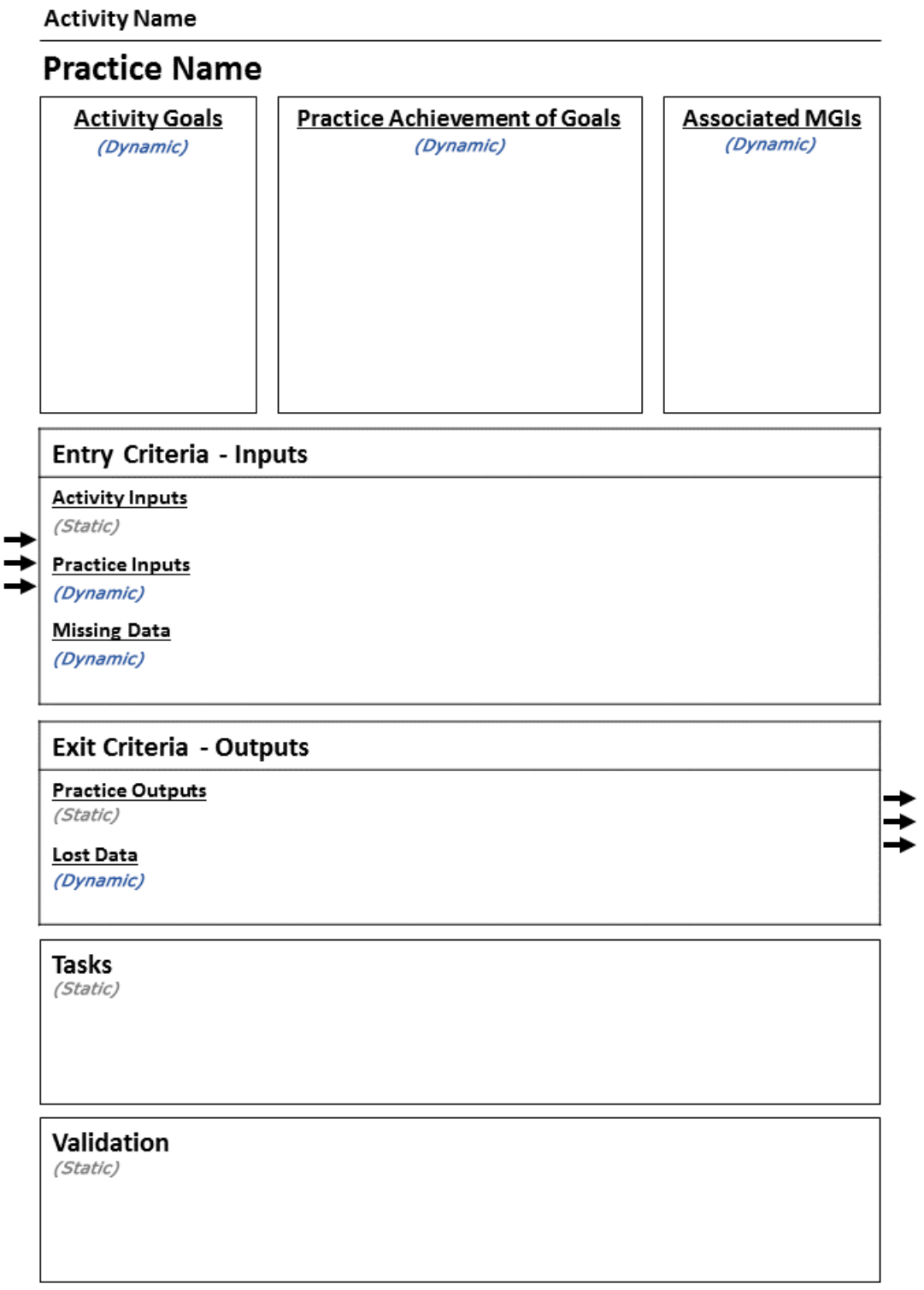


Figure 4.4: The Layout of a Catalog Entry

#### 4.4.1.1 Static Sections of the Catalog Entry

Static sections of a practice's catalog entry represent the information that always applies to a practice regardless of the context.

##### 4.4.1.1.1 Activity Inputs

In order for a practice to be carried out successfully, it may require prior activities in the software process to be completed and may use inputs from these activities to complete its tasks. Entry criteria in the ETVX model represent the criteria that must be satisfied and the actions that must be completed before beginning the tasks of a practice [Radice et al., 1985]. Any entry criteria in the form of previous activities and tasks that must be completed before using a practice are listed in this section. For example, if a developer wants to use the Test Driven Development (TDD) practice to achieve the Construction activity, the Activity Inputs section for the TDD catalog entry would list detailed design as an activity that needs to be completed before attempting TDD.

##### 4.4.1.1.2 Practice Inputs

In addition to inputs from previous activities in the process, a practice may require certain data to exist in order to produce accurate results. An example of Practice Inputs is historical data from previous iterations or projects that may be required if a developer wishes to use the practice at hand to its fullest potential. This section lists all of the required data and the format this data must be in, which will provide the developer with insight into any data transformation that may be required. This section may be left blank if the practice does not require specific information to be available prior to using it.

##### 4.4.1.1.3 Practice Outputs

The Practice Outputs section describes the data that results from completing the tasks of a practice successfully. This information includes the types of outputs, the formats of these

outputs, and any other exit criteria that is context independent. Practice outputs need to be taken into consideration when replacing one practice with another, especially to check for flow compatibility with the practice(s) used in following activities.

#### 4.4.1.1.4 Tasks

The tasks of a practice are the list of actions that indicate what is to be accomplished in order to carry a practice out successfully. The Tasks section of the catalog entry provides the developer with guidance on how to perform a practice, and insight into the amount of effort and experience required to complete it.

#### 4.4.1.1.5 Validation

The Validation section of the catalog entry details how to verify that a practice has been carried out and completed successfully. For example, the Validation section for the TDD catalog entry may state the following [Sanders, 2009]:

- *Each unit test is orthogonal (i.e., independent) to all the others.*
- *Any given behavior is specified in one, and only one test.*
- *There is only one logical assertion per test.*
- *All unit tests are named clearly and consistently.*
- *No unit-tests exist for configuration settings.*

### 4.4.1.2 Dynamic Sections of the Catalog Entry

Each catalog entry contains sections that are determined by the developer's process goals, experience, and current practices in place. These sections are described in detail below:

#### 4.4.1.2.1 Activity Goals

When tailoring a software process using the DASP framework, developers are asked to answer a number of survey questions about their experiences with candidate practices, and the goals they wish to achieve by selecting an alternative practice. A default list of goals is provided containing the goals most common to an activity. Developers are asked to rank these goals based on what they consider to be most valuable from their point of view. New goals can be added and existing goals can be deleted as necessary. The prioritized list of goals is displayed in the Activity Goals section of the catalog entry. An example of a list of goals for the Construction activity may be:

- *Existence of verification*
- *High code quality*
- *Low time consumption*
- *Acceptance testing*

#### 4.4.1.2.2 Practice Achievement of Goals

For each of the goals listed in the Activity Goals section, an indicator of how well the practice at hand achieves each of these goals is displayed. In addition to showing how well this practice meets the developer's goals, this section also shows the performance of the practice at hand in comparison to all other candidate practices for each of the goals listed.

#### 4.4.1.2.3 Associated MGIs

In order to keep track of whether each goal listed in the Activity Goals section is being met, there exist a number of Minimal Guiding Indicators (MGIs) that are constantly monitored. The MGIs being measured are derived from the Practical Software and Systems Measurement

(PSM) Guide's Measurement Selection and Specification Tables [Bailey et al., 2003]. The MGIs can be customized as needed and there can exist multiple MGIs per activity goal.

#### 4.4.1.2.4 Missing Data

The static subsections of the Inputs section explicitly list all of the data required by the practice as inputs from both previous activities in the process and previous iterations or projects. The Missing Data subsection is a dynamic subsection and changes based on the current practice a developer is transitioning from and previous activities in the process. The Missing Data subsection informs the developer of the Practice Inputs and Activity Inputs required by the practice at hand that are not currently available. Data listed in the Missing Data subsection can range from data that can be made available by performing simple transformations, to data that does not exist at all. This subsection will help the developer decide whether transitioning to the practice at hand is a reasonable decision, especially if there is a lot of missing data or if existing data cannot be transformed into a usable format. The Missing Data subsection may be blank if no specific input data is required by a practice or if all data required is available in the correct format. The Missing Data subsection may be blank if one of the following two conditions apply:

1. The practice does not require any specific input data in order to be carried out successfully.

2. All data required is available in the correct format.

#### 4.4.1.2.5 Lost Data

The Lost Data subsection is a subsection of the Outputs section. It provides developers with information crucial to making a successful practice replacement when tailoring their software process. This dynamic subsection describes which input data produced by previous activities and the current practice in place will be lost if the developer chooses to incorporate

the practice at hand practice into his/her software process. The Lost Data subsection may be blank if both of the following two conditions apply:

1. The developer has not used a specific practice to achieve the activity being tailored, such as transitioning from an ad hoc approach to a more structured one, or selecting a practice to use at the beginning of a software project.

2. No data will be lost from previous activities in the process by selecting the practice at hand.

### 4.4.2 Pain Points in Software Process

A pain point in the context of this research is defined as a problem the developer is experiencing with the current software process. Identifying a developer's pain points is a crucial part of customizing a software process to address the specific needs of a developer. Any discomfort or frustration caused by the software process will likely lead to deviating from the process or even abandoning it. Thus, by centering the tailoring of a process around a developer's pain points, the developer's needs are never overlooked, and value from the developer's perspective is the main focus of tailoring. In order to assess the level of pain a developer is experiencing and determine whether or not tailoring the software process will alleviate the developer's pain, multiple levels of pain can be reported. Developers are asked to indicate the level of pain they are experiencing when reporting a pain point. A pain point can vary from slight discomfort to something severe enough to cause the developer to deviate from the process. An example of a pain point might be that a practice is not yielding the desired results, or that performing a specific practice is too time consuming.

Pain is measured on a pain scale from zero to three with zero being "no pain", one being "slight discomfort, I can tolerate the pain for now", two being "painful, I need to adjust the process in order to continue to use it", and three being "severe pain, I was forced to deviate from the process". Both levels 2 and 3 of the pain scale require the process to

be tailored. Level 1 might progress into level 2 if the developer decides that the pain is no longer tolerable.

In order to alleviate pain points, the source of the pain needs to be identified. This will allow us to address the problem more effectively and provide a solution that provides the highest value possible to the developer. Pain points can be detected by one of the following methods:

1. Monitoring the Minimal Guiding Indicators (MGIs).

   If an MGI exceeds a certain threshold that is considered desirable, a pain point is detected and analyzed to determine if tailoring will alleviate the pain.

2. The developer reports a pain point.

   If the developer decides that something in the process is too painful to continue to use without making adjustments or that a practice is not yielding the desired results, this can initiate tailoring the process.

3. The developer deviates from the process.

   Deviation from the process is considered a pain point of level 3 and must be addressed immediately to determine what caused the deviation and make the modifications necessary.

Figure 4.5 shows the format designed to enable software developers to describe a pain point and communicate the level of pain they are experiencing.

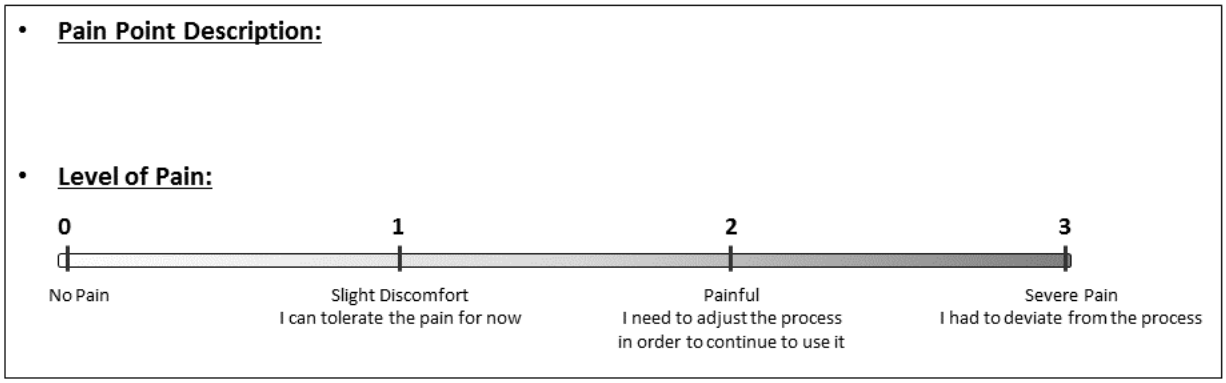


Figure 4.5: The Format Used for Reporting Pain Points

### 4.4.3 Dynamic Tailoring Recommendations Based on Pain Points

This section describes in detail the steps used to arrive at tailoring recommendations based on a developer's pain points. Before evaluating the alternate practices and making a practice replacement recommendation, the source of the pain must be determined in order to identify the software process activity that needs to be tailored. The current practice is replaced with an alternate practice based on the existing practices in place and the software developer's experience.

#### 4.4.3.1 Determining the Source of Pain

Throughout the course of a project while using a software process, a software developer may experience pain or discomfort with certain aspects of the process. In some cases, the pain is tolerable and will subside over time, such as pain determined to be level 1 on the process pain scale, and in other cases the issue is deemed painful enough to require tailoring, such as pain points reported as levels 2 or 3 on the process pain scale.

Reporting painful aspects of the process often proves to be easier than determining the source of the problem. Pain points may originate in practices other than those reported to be painful. In addition, some pain points can be traced back to multiple areas of the software process. Determining the actual source of a pain point allows us to address the

problem more effectively and make adjustments that provide the most value possible to the developer. The first step in the tailoring process is to determine the cause of the pain by finding the software process activity in which the problem occurred. Once the activity has been determined, an alternate practice can be prescribed to alleviate the pain.

The Practical Software and Systems Measurement (PSM) Guide [Bailey et al., 2003] sponsored the US Department of Defense and the US Army "presents a proven approach for defining and implementing an effective measurement process for software and system projects. [Bailey et al., 2003]" The PSM Guide aims to equip managers with significant quantitative information allowing them to make informed decisions that impact project cost, schedule, and technical performance objectives [Bailey et al., 2003]. The effectiveness of the PSM measurement process can be attributed to the fact that the measurement principles described in the guide are compiled from best practices derived from actual experience on government and industry projects [Bailey et al., 2003]. In addition, the PSM measurement process works best when integrated with other management disciplines [Bailey et al., 2003], which makes it an ideal candidate for the selection of MGIs to be measured and monitored for pain points throughout the course of the project. By monitoring specific measures indicative of the success of the activities and practices currently in place in the software process, pain points can be detected and addressed dynamically as they occur.

The PSM guide identifies software project issues and groups them into seven main categories [Bailey et al., 2003]. Figure 4.6 depicts the Issue-Category-Measure Table that "maps Common Issue Areas to related Measurement Categories, and then to measures in each category" [Bailey et al., 2003]. When a pain point is reported and it is found that an MGI has exceeded a specific threshold, we use the Issue-Category-Measure table to trace the MGIs being monitored back to one or more of the seven known project issues. This allows us to determine the issue that caused the pain point. It is possible for multiple candidate issues to be mapped to a single measure or MGI.

| Issue - Category - Measure Mapping | | |
|---|---|---|
| **Common Issue Area** | **Measurement Category** | **Measures** |
| Schedule and Progress | Milestone Performance | Milestone Dates<br>Critical Path Performance |
| | Work Unit Progress | Requirements Status<br>Problem Report Status<br>Review Status<br>Change Request Status<br>Component Status<br>Test Status<br>Action Item Status |
| | Incremental Capability | Increment Content - Components<br>Increment Content - Functions |
| Resources and Cost | Personnel | Effort<br>Staff Experience<br>Staff Turnover |
| | Financial Performance | Earned Value<br>Cost |
| | Environment and Support Resources | Resource Availability<br>Resource Utilization |
| Product Size and Stability | Physical Size and Stability | Database Size<br>Components<br>Interfaces<br>Lines of Code<br>Physical Dimensions |
| | Functional Size and Stability | Requirements<br>Functional Change Workload<br>Function Points |
| Product Quality | Functional Correctness | Defects<br>Technical Performance |
| | Supportability - Maintainability | Time to Restore<br>Cyclomatic Complexity<br>Maintenance Actions |
| | Efficiency | Utilization<br>Throughput<br>Timing |
| | Portability | Standards Compliance |
| | Usability | Operator Errors |
| | Dependability - Reliability | Failures<br>Fault Tolerance |
| Process Performance | Process Compliance | Reference Model Rating<br>Process Audit Findings |
| | Process Efficiency | Productivity<br>Cycle Time |
| | Process Effectiveness | Defect Containment<br>Rework |
| Technology Effectiveness | Technology Suitability<br>Impact<br>Technology Volatility | Requirements Coverage<br>Technology Impact<br>Baseline Changes |
| Customer Satisfaction | Customer Feedback | Survey Results<br>Performance Rating |
| | Customer Support | Requests for Support<br>Support Time |

Figure 4.6: PSM's Issue-Category-Measure Table [Bailey et al., 2003]

In order to replace the painful MEP with an alternate MEP, the MSA in which the issue originated must be identified. We mapped PSM's common project issue areas to the PCSE MSAs to simplify the determination of the activity in which the pain originated as shown in Figure 4.7.

Once the MSA in which tailoring will occur is decided, the MEP currently in place needs to be replaced with an alternate MEP. In order to select the best-fitting alternate MEP, all possible MEPs are evaluated. The methodology described next can be used to tailor a software process based on pain points. It can also be used for the evaluation of multiple practices prior to starting a new project to select the most suitable practice among a number of possible practices.

#### 4.4.3.2 Evaluation of Alternative Practices

The DASP framework provides developers with a practice selection process that helps evaluate candidate practices, and makes recommendations based on attributes and features that the developer values most in a given context. It allows developers to make informed decisions regarding the alternate practices that can be introduced into their process. The DASP allows the developer to:

1. Explore the cost and benefit of introducing each of the candidate practices into the existing software process.

2. Decide among the available practices based on their value from the developer's point of view and the cost of introducing them into the process.

##### 4.4.3.2.1 Using the Developer's Input to Guide the Evaluation

Each software developer possesses a unique set of skills and experiences. These experiences determine the developer's knowledge of different software practices and provide insight into the existence of any useful historical data that may aid in the accuracy of the results yielded

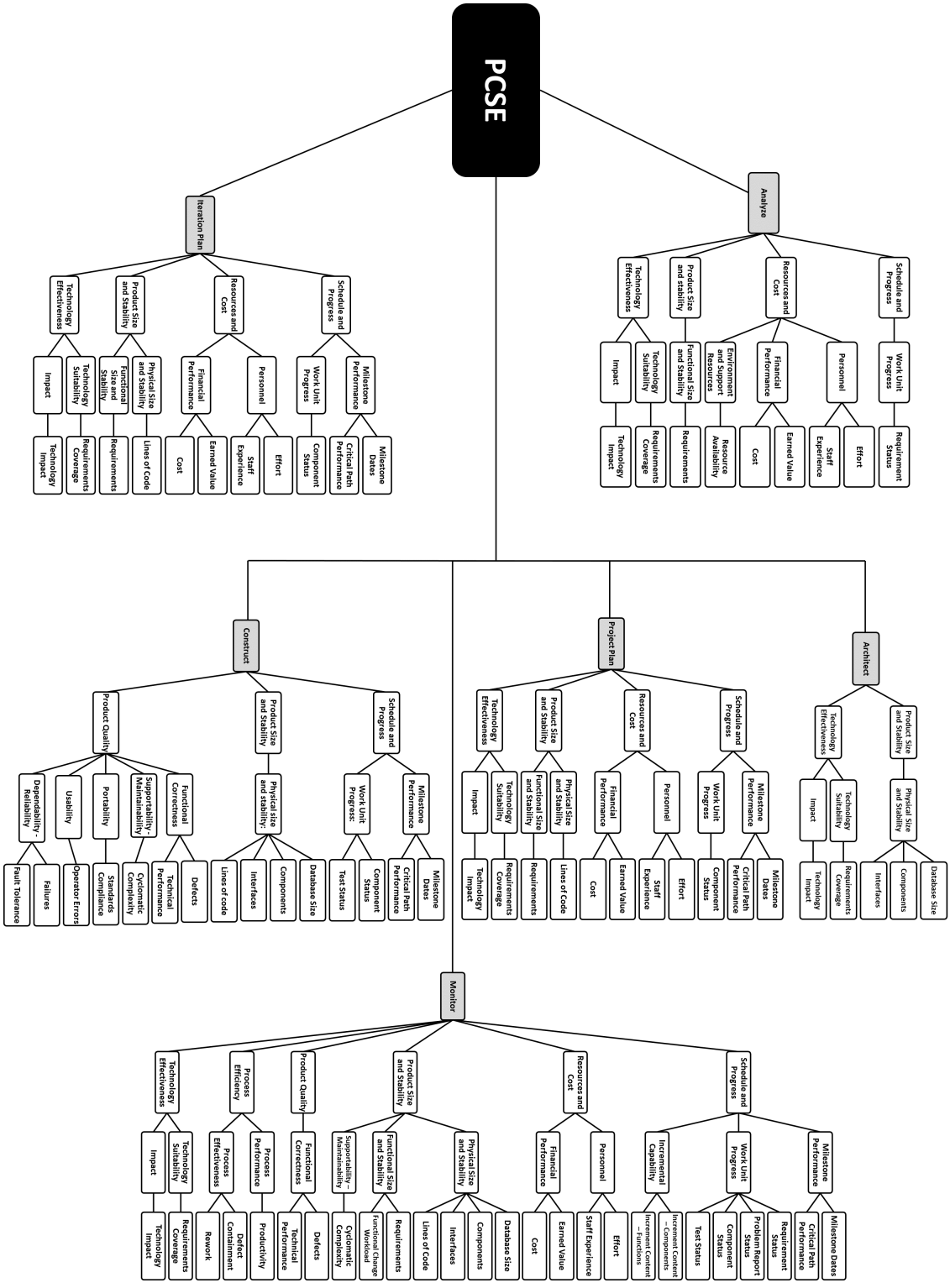


Figure 4.7: Mapping PSM's Common Issue Areas to PCSE's MSAs

by specific software practices. Before evaluating candidate practices and introducing them into the software process, it is important to gather information regarding the developer's experience and comfort level with the candidate practices. In addition, information pertaining to the nature of the project at hand and the developer's expectations from the personal software process in general and from the practice that will be introduced into the process in specific can be gathered. The information provided by the software developer allows us to assess the practices more accurately and provide the developer with a more satisfactory process. Information is gathered in the form of a minimal survey at the beginning of each project.

Once the painful MSA is detected, the MEP currently being used to implement that activity needs to be replaced with an alternate MEP that should alleviate the pain. The candidate MEPs are retrieved from the MEP catalog and listed for the developer's input. The developer's input regarding his/her expectations from the MSA as well as important information about each of the candidate MEPs is recorded. The developer has the option to add or remove candidate practices from the default MEP list as necessary. The following information is collected from the developer:

1. The developer's prioritized list of MSA goals based on what s/he values most. For each MSA, a default list of the most common goals is provided for the developer to rank. The developer can customize the MSA goals as needed by adding or deleting goals.

2. Proficiency and experience level with each of the candidate practices ranked on a 5-point Likert scale from 1 being "no experience", to 5 being "very experienced".

3. The existence of historical data for each of the practices. Historical data can refer to data gathered from previous iterations in the current project or existing data from previous projects.

4. The possibility of performing data transformation on the historical data available to inputs required by each of the candidate practices.

5. The inputs required by each of the candidate practices that will be missing upon the selection of one these practices to replace the current practice in the software process.

6. The output data of the current practice that will be lost upon the selection of each of the candidate practices to replace the current practice in the software process.

#### 4.4.3.2.2 Measuring Practice Achievement of the Developer's Goals

After an initial collection of developer input, the developer's priorities in terms of goals are now known. The next step is to evaluate how well each of the candidate practices achieves these goals. The Goal Question Metric (GQM) paradigm [Basili, 1992] allows for goals to be evaluated in a quantitative way. An important feature of the GQM paradigm is its support of both objective and subjective metrics. [Basili, 1992] explains that in the context of GQM, "subjective metrics represent an estimate of extent or degree in the application of some technique or a classification or qualification of problem or experience." Subjective metrics are usually based on a nominal or ordinal scale since there is no exact measurement. In DASP, we use an ordinal scale to represent priority among a developer's MSA goals. The following steps explain how the GQM paradigm is applied to quantify the effectiveness of each of the candidate practices at achieving the developer's goals:

1. Create a set of activity goals based on what the developer values most. Since developer goals are previously listed and prioritized in the developer survey, the existing list of prioritized goals is used for this step of applying the GQM paradigm.

   For example, the following list represents the default goals for the Construction MSA prioritized based on their importance to a given developer:

   - *Quality of code.*
   - *Test coverage.*
   - *Low time consumption.*

- *Low learning curve*
- *Acceptance testing*

2. Derive a set of questions from each goal that allow us to assess the achievement of that goal. The number of questions can vary from one to many per goal as necessary. The developer can add or remove questions as necessary.

   For example, the "acceptance testing" goal has a single question linked to it: Does the MEP provide acceptance testing?

   *1 - Yes*

   *0 - No*

3. Associate a set of data with each of the questions to enable its quantification. Since the goal data is subjective in this case and depends mainly on the developer's point of view, the answer to each question is quantified by ranking each practice's performance on an interval scale.

   For example, when ranking the Construction MEPs' achievement of the question "how well does this MEP achieve test coverage?", 0 implies that there is low or no code coverage, 1 implies that there is moderate code coverage, and 2 implies that there is full or near full code coverage. The numbers on the interval scale are later normalized to range from 0 to 1.

#### 4.4.3.2.3 Evaluating Candidate Practices and Making Recommendations Based on the Developer's Input

After defining the developer's goals for a given MSA and formulating a number of questions that will assess whether these goals are being met, we must evaluate each of the candidate MEPs against these goals to determine the best-in-class performer and make a recommendation that will best suit the developer's needs.

The Multiple Attribute Utility (MAU) approach is a mathematical approach to ranking and selection that allows for the evaluation of multiple performance measures and the selection of the best alternative from a set of candidate alternatives.

The following steps describe how the MAU approach is applied in the DASP framework to rank and score each of the candidate MEPs and find the recommendation that will best replace the current MEP and alleviate the developer's pain:

1. Establish a set of evaluation criteria. The GQM paradigm used in the previous stage of DASP carefully defines the evaluation criteria since this set of criteria will be used to evaluate how well each of the MEPs performs. The set of goals defined for each MSA is the first level of evaluation criteria. The questions that measure the achievement of these goals are considered a second level of evaluation criteria used for the MAU evaluation.

2. Define how each candidate MEP will be scored against the evaluation criteria by selecting a scoring function. Since most of the evaluation taking place is subjective, a constructed scale such as function ui shown in the example below is used [Brown, 2007]:

   $u_i(a_i) = 0$ if a product does not meet evaluation criteria $a_i$

   $u_i(a_i) = 1$ if a product partially meets evaluation criteria $a_i$

   $u_i(a_i) = 2$ if a product fully meets evaluation criteria $a_i$

   The scoring function must then be normalized based on MAU convention so that the scores range from 0 to 1. The normalized function in the previous example produces the following function [Brown, 2007]:

   $u_i(a_i) = 0$ if a product does not meet evaluation criteria $a_i$

   $u_i(a_i) = .5$ if a product partially meets evaluation criteria $a_i$

   $u_i(a_i) = 1$ if a product fully meets evaluation criteria $a_i$

This scoring method is not limited to a specific number of values. The previous example contains a discrete set of three values, but smaller or larger discrete sets can be used as necessary.

3. Assign weights to the evaluation criteria. The Reference Comparison method was selected to assign weights to the evaluation criteria (MSA goals). The Reference Comparison method can be used with any number of evaluation criteria, and uses the importance or priority of the evaluation criteria to determine the relative weights of each of the criteria. This is especially convenient since the evaluation criteria has already been prioritized in pervious stages of the DASP framework, and is subjective in nature, so only relative priority is known. The number of criteria (MSA goals) is used as the reference and assigned to the goal with the highest priority. The remaining goals are ranked in descending order, then all values are normalized. The following example demonstrates how 3 goals would be ranked:

   3 = *the goal with the highest priority, this is considered the "reference criterion".*

   *2 = the goal with the second highest priority, this is less important than the "reference criterion".*

   *1 = the goal with the lowest priority. this is less important than all previous goals listed.*

   The number of goals can vary based on the MSA and the developer's input. The number of goals will always be used as the reference criterion. After the goals are ranked, the values are normalized so that the sum of all values is 1. The ranks are summed, and each is divided by the sum of all ranks to determine the weights of the criteria. In the previous example, the sum of all ranks is 6. Tor normalize the values we divide each of them by 6. The weights assigned to the three goals are as follows:

   *3/6 = 0.5 (the goal with the highest priority)*

   *2/6 = 0.33 (the goal with the second highest priority)*

*1/6 = 0.67 (the goal with the lowest priority)*

Within each of the MSA goals, the Reference Comparison method is also used to assign weights to the answers of the questions that are used to assess how well the goal is achieved by an MEP. If multiple questions exist for a single goal, the weight of that goal is divided equally among those questions.

4. Calculate the overall score for each of the candidate MEPs being evaluated. Now that each evaluation criterion (MSA goal) has a weight assigned to it, the questions that assess the MEP achievement of that goal have weights assigned to them, and the answers to each of the questions also have weights assigned to them, the values associated with each of the MEPs are summed up and compared. The candidate MEP with the highest score is recommended to the developer as a replacement for the current MEP. To demonstrate how the sums are calculated for each MEP, the additive utility function with three evaluation criteria $c_1$, $c_2$, and $c_3$, each with two questions is shown:

   $u(c_1, c_2, c_3) = (qw_1 * a_1 + qw_1 * a_2) + (qw_2 * a_3 + qw_2 * a_4) + (qw_3 * a_5 + qw_3 * a_6)$

   Where:

   - $u$ is the overall score for a given MEP, based on three evaluation criteria: $c_1$, $c_2$, and $c_3$.
   - $c_1$,$c_2$, and $c_3$ are the evaluation criteria (MSA goals) that the MEP is being evaluated against.
   - $qw_1$ is the weight assigned to each of the two questions for the highest ranked criterion, $c_1$. $qw_1$ is calculated by dividing the weight of $c_1$ by the number of questions for $c_1$, which is 2 in this example.
   - $a_1, .., a_6$ are the weights of the answers of the six questions (two per evaluation criterion) for the given MEP.

- $qw_2$ is the weight assigned to each of the two questions for the second ranked criterion, $c_2$. $qw_2$ is calculated by dividing the weight of $c_2$ by the number of questions for $c_2$, which is 2 in this example.
- $qw_3$ is the weight assigned to each of the two questions for the third ranked criterion, $c_3$. $qw_3$ is calculated by dividing the weight of $c_3$ by the number of questions for $c_3$, which is 2 in this example.

After the additive utility functions have been calculated for all candidate MEPs, we can now easily compare the scores and determine the best candidate for replacing the current MEP in the process.

## 4.5 Summary

The DASP framework guides a software developer through tailoring a software process in a custom way that best fits his/her needs. The tailoring process begins with correctly identifying the source of pain, and ends with recommending a software practice to replace the painful one after analyzing and evaluating all candidate software practices. Since the DASP framework focuses on value from the developer's point of view and aims to offer solutions customized to best fit a developer's needs, the developer's input regarding his/her goals and experience play an important role in finding a practice that will address his/her pain points. The DASP framework incorporates a number of well-known quantification and evaluation techniques such as the GQM and MAU approaches to carefully evaluate the candidate practices at hand and make a recommendation for alleviating the developer's pain.

# Chapter 5

# Framework Samples

This chapter demonstrates how the DASP framework operates and is able to provide software developers with recommendations that best fits their needs based on their experiences. The detailed scenarios presented next show how some of the common pain points in various MSAs are addressed and alleviated with the recommendations made by the DASP framework.

## 5.1 Framework Sample 1 - Tailoring the Construction MSA Based on a Pain Point

For this demonstration, the software developer at hand is a student using a minimal version of PCSE. For the Minimal Viable Process (MVP) the student was using the Construction MSA was being performed in an ad hoc fashion. After a few iterations the student reported a pain point in the software process.

### 5.1.1 Reporting the Pain Point

The software developer reports that there are too many defects in the code. 5.1. depicts the pain point that was reported:

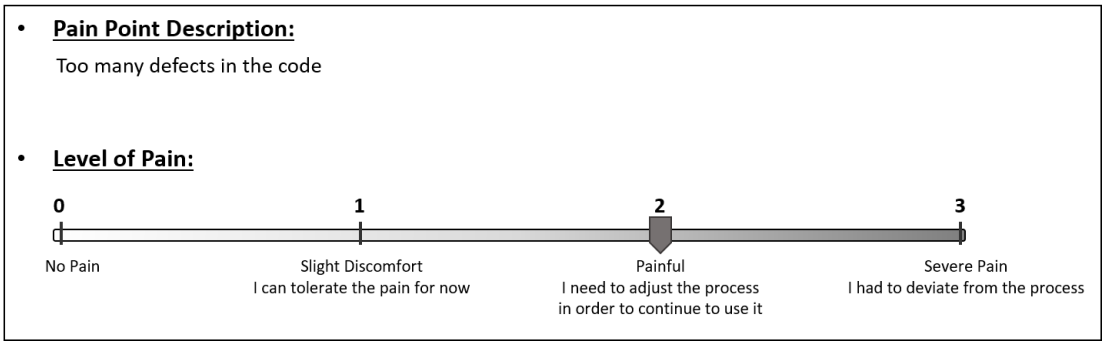


Figure 5.1: Pain Point Reported by the Software Developer

### 5.1.2 Determining the Source of Pain

The first step in alleviating the pain a software developer is experiencing is to determine the source of pain. By examining the PCSE/PSM Common Issue Area Map, it is determined that the Construction MSA is most likely the source of the pain. This leads us to take a look at the Construction MEP being used. Since the software developer is performing the Construction MSA in an ad hoc fashion, no MEP is currently in place, and the available MEPs for the Construction MSA need to be evaluated.

### 5.1.3 Gathering Developer Input

Before evaluating the candidate MEPs, the software developer is asked to fill out a minimal survey regarding the painful MSA. This provides us with the developer's goals and expectations from the MEP that will be introduced into the process, as well as the developer's previous experiences with the candidate MEPs. Figure5.2 shows the completed survey for the Construction MSA completed by the software developer.

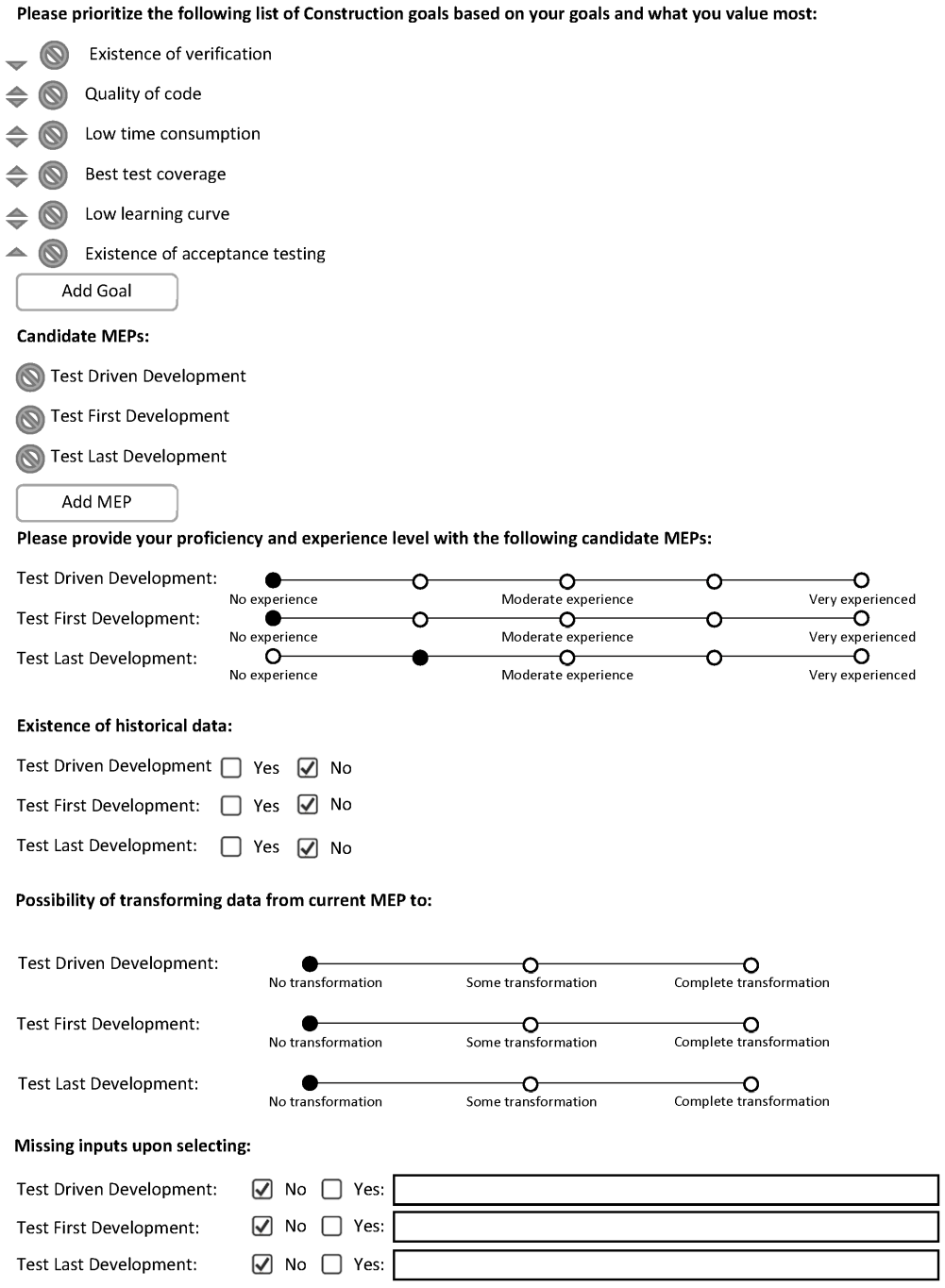


Figure 5.2: Developer Survey for the Construction MSA

### 5.1.4 Applying the GQM Paradigm to Refine Evaluation Criteria

As a first step to defining and refining the evaluation criteria, the software developer's prioritized list of MSA goals is retrieved from the developer survey. The following list of goals has been provided for the Construction MSA:

*Existence of verification*

*Quality of code*

*Low time consumption*

*Best test coverage*

*Low learning curve*

*Existence of acceptance testing*

Now that the MSA goals are available, questions that assess an MEP's achievement of the goals are derived. For the listed goals, the following questions and possible answers have been defined:

**Goal:** Existence of verification

**Question:** Does this MEP provide a means of code verification?

Yes

No

**Goal:** Quality of code

**Question:** Which of the options below best describes the quality of code typically produced by this MEP?

Best code quality

Good code quality

Poor code quality

**Goal:** Low time consumption

**Question:** Which of the options below best describes the amount of time this MEP consumes in comparison to other Construction MEPs?

Low time consumption

Moderate time consumption

Time consuming

**Goal:** Best test coverage

**Question:** In terms of testing, what level of code coverage does this MEP offer?

Full/near full code coverage

Moderate code coverage

Low/no code coverage

**Goal:** Low learning curve

**Question:** What is the learning curve associated with this MEP?

Low learning curve

Moderate learning curve

High learning curve

**Goal:** Existence of acceptance testing

**Question:** Does this MEP provide the developer with an option for acceptance testing?

Yes

No

The final step of applying the GQM paradigm is to associate a set of data with the answers to each of the questions. Since the answers to the above questions are subjective, an interval scale is used. The answers are ranked as follows:

**Goal:** Existence of verification

**Question:** Does this MEP provide a means of code verification?

1 - Yes

0 - No

**Goal:** Quality of code

**Question:** Which of the options below best describes the quality of code typically produced by this MEP?

3 - Best code quality

2 - Good code quality

1 - Poor code quality

**Goal:** Low time consumption

**Question:** Which of the options below best describes the amount of time this MEP consumes in comparison to other Construction MEPs?

3 - Low time consumption

2 - Moderate time consumption

1 - Time consuming

**Goal:** Best test coverage

**Question:** In terms of testing, what level of code coverage does this MEP offer?

2 - Full/near full code coverage

1 - Moderate code coverage

0 - Low/no code coverage

**Goal:** Low learning curve

**Question:** What is the learning curve associated with this MEP?

3 - Low learning curve

2 - Moderate learning curve

1 - High learning curve

**Goal:** Existence of acceptance testing

**Question:** Does this MEP provide the developer with an option for acceptance testing?

1 - Yes

0 - No

Next the ranks are normalized to range between 0 and 1:

**Goal:** Existence of verification

**Question:** Does this MEP provide a means of code verification?

1.00 - Yes

0.00 - No

**Goal:** Quality of code

**Question:** Which of the options below best describes the quality of code typically produced by this MEP?

1.00 - Best code quality

0.67 - Good code quality

0.33 - Poor code quality

**Goal:** Low time consumption

**Question:** Which of the options below best describes the amount of time this MEP consumes in comparison to other Construction MEPs?

1.00 - Low time consumption

0.67 - Moderate time consumption

0.33 - Time consuming

**Goal:** Best test coverage

**Question:** In terms of testing, what level of code coverage does this MEP offer?

1.00 - Full/near full code coverage

0.50 - Moderate code coverage

0.00 - Low/no code coverage

**Goal:** Low learning curve

**Question:** What is the learning curve associated with this MEP?

1.00 - Low learning curve

0.67 - Moderate learning curve

0.33 - High learning curve

**Goal:** Existence of acceptance testing

**Question:** Does this MEP provide the developer with an option for acceptance testing?

1.00 - Yes

0.00 - No

### 5.1.5 Evaluating Candidate MEPs and Making a Recommendation

After defining the evaluation criteria, the candidate MEPs for the Construction MSA are evaluated against these criteria. The MEPs compared and evaluated were: Test Driven Development, Test First Development, Test Last Development, and ad hoc development (the method currently in place). Figure 5.3 details the evaluation process using the MAU approach.

| Goal | Question | Weight | | TDD | TFD | TLD | Ad Hoc |
|---|---|---|---|---|---|---|---|
| Existence of verification | Does this MEP provide a means of code verification? | | 0.29 | 1.00 | 1.00 | 1.00 | 0.00 |
| | Yes | 1.00 | | | | | |
| | No | 0.00 | | | | | |
| | Score | | | 0.29 | 0.29 | 0.29 | 0.00 |
| Quality of code | Which of the options below best describes the quality of code typically produced by this MEP? | | 0.24 | 1.00 | 0.67 | 0.67 | 0.33 |
| | Best code quality | 1.00 | | | | | |
| | Good code quality | 0.67 | | | | | |
| | Poor code quality | 0.33 | | | | | |
| | Score | | | 0.24 | 0.16 | 0.16 | 0.08 |
| Low time consumption | Which of these options best describes the amount of time this MEP consumes in comparison to other Construction MEPs? | | 0.19 | 0.33 | 0.67 | 0.67 | 1.00 |
| | Low time consumption | 1.00 | | | | | |
| | Moderate time consumption | 0.67 | | | | | |
| | Time consuming | 0.33 | | | | | |
| | Score | | | 0.06 | 0.13 | 0.13 | 0.19 |
| Best test coverage | In terms of testing, what level of code coverage does this MEP offer? | | 0.14 | 1.00 | 0.50 | 0.50 | 0.00 |
| | Full/near full code coverage | 1.00 | | | | | |
| | Moderate code coverage | 0.50 | | | | | |
| | Low/no code coverage | 0.00 | | | | | |
| | Score | | | 0.14 | 0.07 | 0.07 | 0.00 |
| Low learning curve | What is the learning curve associated with this MEP? | | 0.10 | 0.33 | 0.67 | 0.67 | 1.00 |
| | Low learning curve | 1.00 | | | | | |
| | Moderate learning curve | 0.67 | | | | | |
| | High learning curve | 0.33 | | | | | |
| | Score | | | 0.03 | 0.06 | 0.06 | 0.10 |
| Acceptance testing | Does this MEP provide the developer with an option for acceptance testing? | | 0.05 | 0.00 | 1.00 | 0.00 | 0.00 |
| | Yes | 1.00 | | | | | |
| | No | 0.00 | | | | | |
| | Score | | | 0.00 | 0.05 | 0.00 | 0.00 |
| Total Score | | | | 0.76 | 0.75 | 0.71 | 0.37 |

Figure 5.3: The Evaluation of Candidate MEPs to Find a Recommendation

The column titled "Goal" lists the software developer's goals starting with the highest-priority goal at the top of the list and ending with the lowest-priority goal at the bottom. The

"Question" column contains the questions that were defined to further refine the evaluation criteria, as well as the possible answers to these questions. In this example, the achievement of each goal is evaluated by a single question. The "Weight" column is divided into two sections. The section on the left contains the ranks of the answers defined on an interval scale and then normalized to range between 0 and 1. 0 represents no achievement of the goal and 1 represents full achievement of the goal. The section on the right contains the weights of each goal. The number of goals is used as the reference criterion, and goals are assigned weights in descending order starting with the highest-priority goal and ending with the lowest-priority goal. In this example the number of goals is 6, so the goal of "existence of verification" is assigned a weight of 6, whereas the goal of "acceptance testing" is assigned a weight of 1. According to the Reference Comparison approach described in Chapter 4, the weights are then summed up, and each weight is divided by the sum of all weights so that the sum of all values is 1. The next four columns represent the candidate MEPs. For each goal, the MEP's answer weight can be seen as well as the final weight that the MEP receives for that goal after multiplying the answer weight by the goal weight. Each MEP's total is calculated by summing up its score for each goal.

It can be seen that in this example, based on the software developer's goals and priorities, the recommended MEP was the Test Driven Development practice. Test First Development comes next, followed by Test Last Development, and finally the ad hoc approach the student reported as painful. Figure5.4 shows an example of a the catalog entry for the Test Driven Development MEP. The developer may want to view the catalog entries for both Test Driven Development and Test First Development since the scores were very close.

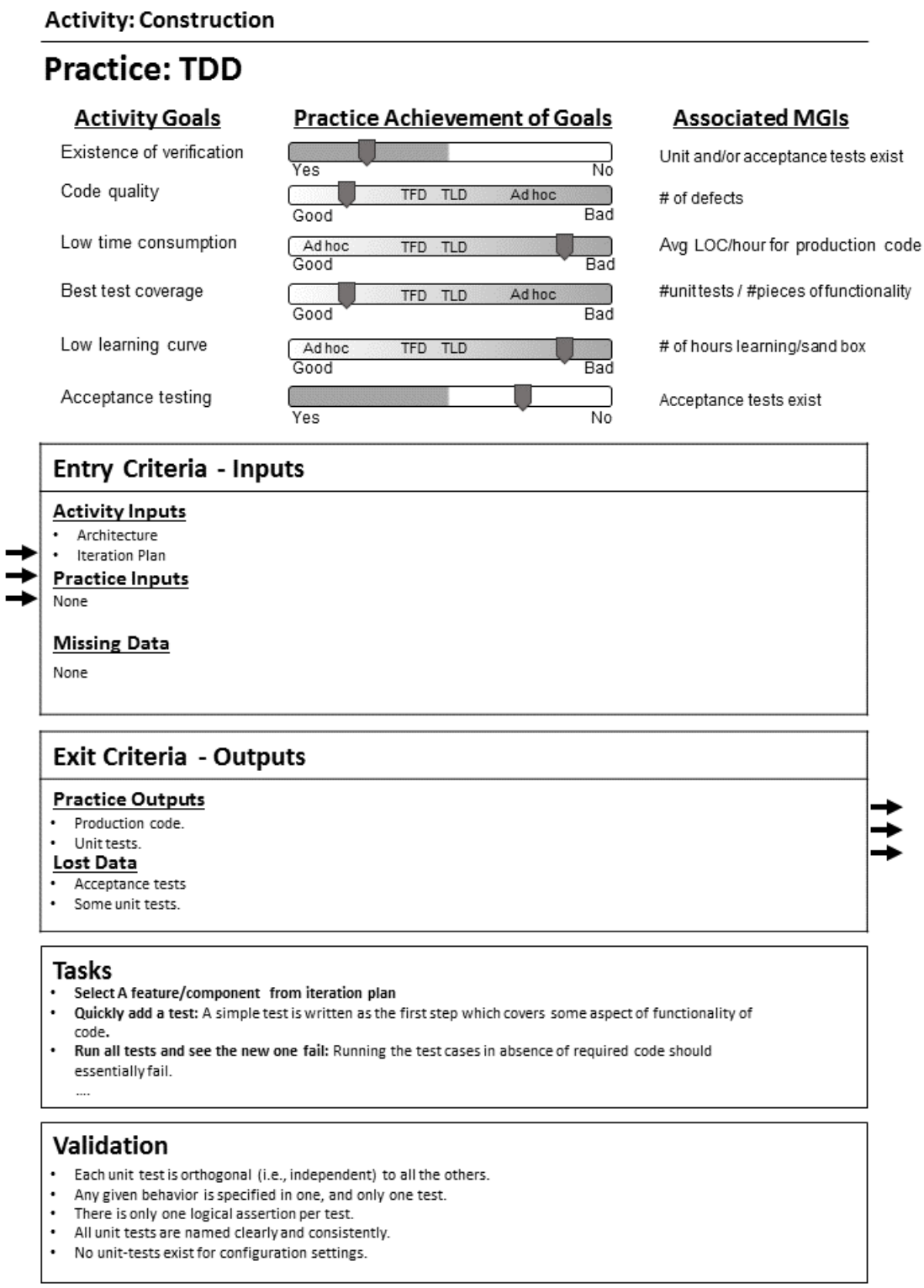


Figure 5.4: The Catalog Entry for the TDD MEP

In the TDD catalog entry above, the Activity Goals are retrieved dynamically from the Developer Survey. The Practice Achievement of Goals section is populated dynamically from the MAU evaluation, and shows how each of the candidate MEP compares to the MEP at hand for each of the MSA goals.

## 5.2 Framework Sample 2 - Tailoring the Estimation MEP Based on a Pain Point

The software developer in this example is a student that was using SISE, an estimation technique developed at Auburn University that offers developers with a light-weight alternative for estimating the amount of time required to build a software component. The Estimation MSA differs from the Construction MSA in that the accuracy of estimations rely heavily on the existence of historical data that provides information about a developer's performance and productivity in the past. In this sample, the software developer had used SISE for multiple iterations and has historical data from SISE.

### 5.2.1 Reporting the Pain Point

After completing a few iterations of the project, the software developer reported the pain point shown in Figure 5.5 below.

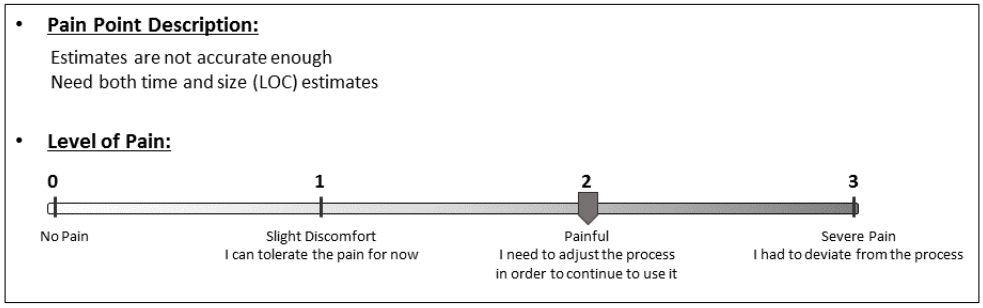


Figure 5.5: Pain Point Reported by the Software Developer

### 5.2.2 Gathering Developer Input

Before evaluating the candidate MEPs, the software developer files out the Developer Survey to provide us with his/her goals and expectations from the MEP that will be introduced into the process, as well as any relevant information regarding historical data and previous experience with the candidate MEPs. Figure 5.6 shows the completed survey.

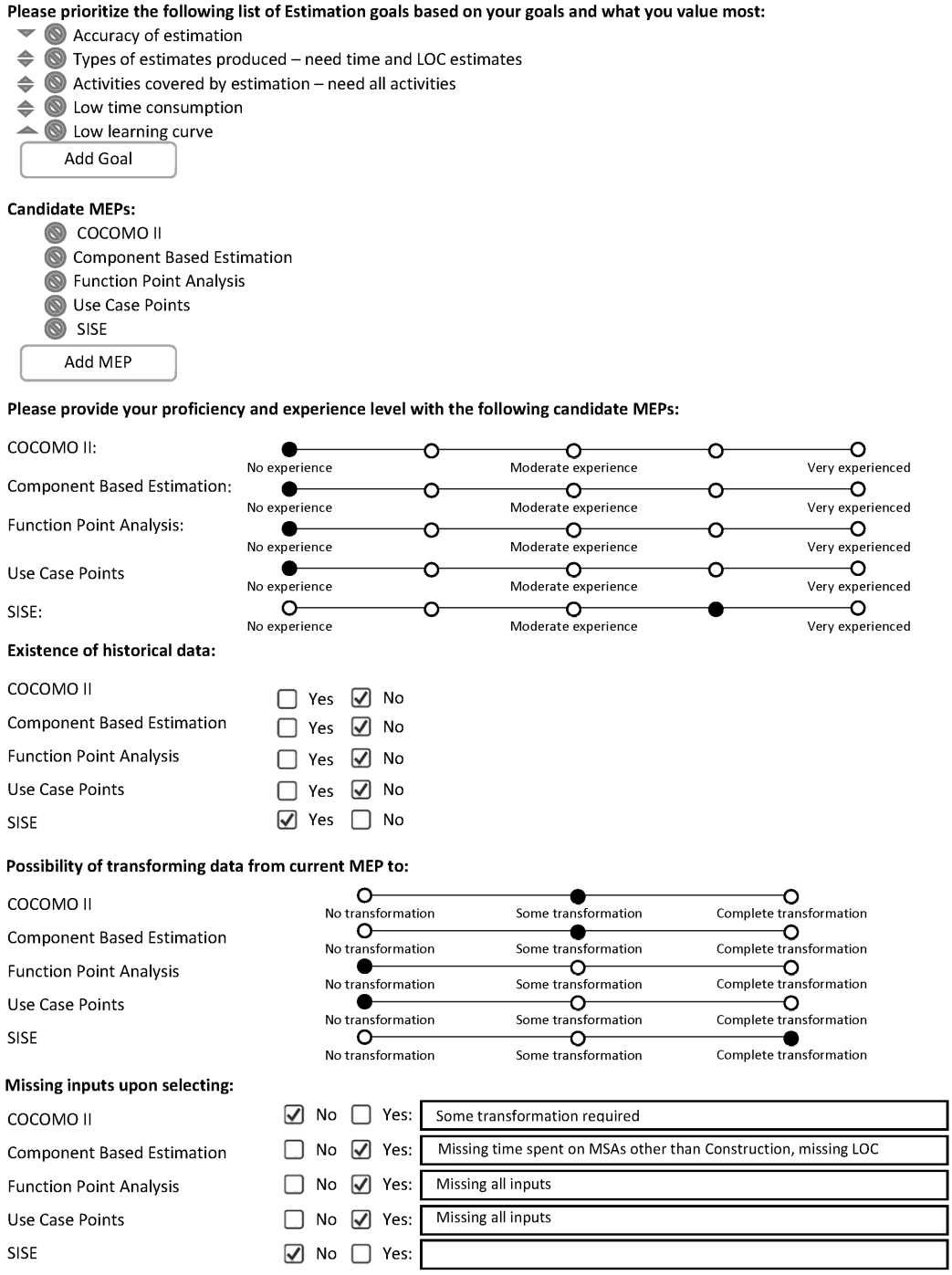


Figure 5.6: Developer Survey for the Estimation MSA

### 5.2.3 Applying the GQM Paradigm to Refine Evaluation Criteria

After retrieving the developer's goals from the Developer Survey, the GQM paradigm is used to refine the estimation criteria by defining questions that assess each MEP's achievement of the goals. In this example, it can be observed that many of the goals are evaluated by asking multiple questions. It can also be observed that it is possible to have common questions across multiple goals. A list of goals and associated questions for the Estimation MSA are shown below. This list contains the ranks of the answers to each question after they were normalized to range between 0 and 1.

**Goal:** Accuracy of estimation

**Question:** Does historical data exist?

1.00 - Yes

0.00 - No

**Question:** What is the amount of data transformation required?

1.00 - Does not require any transformation (same practice)

0.67 - Simple data transformation (all inputs are available)

0.33 - Complex data transformation (only some inputs available)

0.00 - Data cannot be transformed/No historical data exists

**Question:**Does estimation method cover all activities?

1.00 Yes

0.00 No

**Goal:** Types of estimates produced

**Question:** Does estimation MEP offer required estimates?

1.00 - Yes

0.00 - No

**Goal:** Activities covered by estimation

**Question:** Does the estimation MEP cover required activities only?

1.00 - Yes

0.00 - No (covers other/additional activities of the process)

**Goal:** Low time consumption

**Question:** What amount of training is required to become proficient with MEP?

1.00 - None (developer already familiar with practice

0.67 - Simple training required (practice is easy to learn)

0.33 - Advanced (professional) training required

**Question:** What is the amount of data collection and overall effort required by MEP?

1.00 - None

0.75 - Simple data collection (beginning and end of each project)

0.50 - Moderate data collection (end of each iteration)

0.25 - Continuous data collection (each activity)

**Question:** Are any supporting tools for automation available?

1.00 - Tools available and minimum effort required form developer

0.50 - Some operations are automated but not all

0.00 - No automation/tool support available

**Goal:** Low learning curve

**Question:** What amount of training is required to become proficient with MEP?

1.00 - None (developer already familiar with practice)

0.67 - Simple training required (practice is easy to learn)

0.33 - Advanced (professional) training required

### 5.2.4 Evaluating Candidate MEPs and Making a Recommendation

Figure 5.7 shows the evaluation process of the candidate MEPs. Based on the developer's goals and priorities the recommended MEP is Component Based Estimation.

| Goal | Questions | Question Weights | | Goal Weight | COCOMO II | CBE | FPA | UCP | SISE |
|---|---|---|---|---|---|---|---|---|---|
| Accuracy of estimation | Does historical data exist? | | 0.11 | 0.33 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| | Yes | 1.00 | | | | | | | |
| | No | 0.00 | | | | | | | |
| | What is the amount of data transformation required? | | 0.11 | | 0.67 | 0.33 | 0.00 | 0.00 | 1.00 |
| | Does not require any transformation (same practice) | 1.00 | | | | | | | |
| | Simple data transformation (all inputs are available) | 0.67 | | | | | | | |
| | Complex data transformation (only some inputs available) | 0.33 | | | | | | | |
| | Data cannot be transformed/No historical data exists | 0.00 | | | | | | | |
| | Does estimation method cover all activities? | | 0.11 | | 1.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| | Yes | 1.00 | | | | | | | |
| | No | 0.00 | | | | | | | |
| | Score | | | | 0.30 | 0.26 | 0.11 | 0.11 | 0.22 |
| Types of estimates produced | Does estimation method offer required estimates? | | 0.27 | 0.27 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| | Yes | 1.00 | | | | | | | |
| | No | 0.00 | | | | | | | |
| | Score | | | | 0.00 | 0.27 | 0.00 | 0.00 | 0.00 |
| Activities covered by estimation | Does the estimation method cover required activities only | | 0.20 | 0.20 | 1.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| | Yes | 1.00 | | | | | | | |
| | No (covers other/additional activities of the process) | 0.00 | | | | | | | |
| | Score | | | | 0.20 | 0.20 | 0.00 | 0.00 | 0.00 |
| Low time consumption | What amount of training is required to become proficient with MEP? | | 0.04 | 0.13 | 0.67 | 0.67 | 0.33 | 0.67 | 1.00 |
| | None (developer already familiar with practice) | 1.00 | | | | | | | |
| | Simple training required (practice is easy to learn) | 0.67 | | | | | | | |
| | Advanced (professional) training required | 0.33 | | | | | | | |
| | What is the amount of data collection and overall effort required by MEP? | | 0.04 | | 0.75 | 0.25 | 0.5 | 0.5 | 0.75 |
| | None | 1.00 | | | | | | | |
| | Simple data collection (beginning and end of each project) | 0.75 | | | | | | | |
| | Moderate data collection (end of each iteration) | 0.50 | | | | | | | |
| | Continuous data collection (each activity) | 0.25 | | | | | | | |
| | Are any supporting tools for automation available? | | 0.04 | | 1 | 0.5 | 1 | 1 | 0.5 |
| | Tools available and minimum effort required form developer | 1.00 | | | | | | | |
| | Some operations are automated but not all | 0.50 | | | | | | | |
| | No automation/tool support available | 0.00 | | | | | | | |
| | Score | | | | 0.11 | 0.06 | 0.08 | 0.10 | 0.10 |
| Low learning curve | What amount of training is required to become proficient with MEP? | | 0.07 | 0.07 | 0.67 | 0.67 | 0.33 | 0.67 | 0.67 |
| | None (developer already familiar with practice) | 1.00 | | | | | | | |
| | Simple training required (practice is easy to learn) | 0.67 | | | | | | | |
| | Advanced (professional) training required | 0.33 | | | | | | | |
| | Score | | | | 0.04 | 0.04 | 0.02 | 0.04 | 0.04 |
| Total Score | | | | | 0.65 | 0.83 | 0.21 | 0.25 | 0.37 |

Figure 5.7: The Evaluation of Candidate MEPs to Find a Recommendation

In the evaluation above, the "Goal" column contains the prioritized list of the software developer's goals, with the highest-priority goal "accuracy of estimation" at the top of the list, and the lowest-priority goal "low learning curve" at the bottom of the list. The "Questions" column contains the questions that refine the evaluation criteria. Unlike the previous example where each goal had a single question to assess it, it can be seen that many of the goals have multiple questions to assess them in this example. the "Question Weights" column is split into two sections. The left section displays the answer ranks after they have been normalized to range between 0 and 1. The right section displays the weight of each question, as the

total weight for each goal is divided equally among all questions used to assess it. The "Goal Weight" column contains the final weights assigned to each goal. The values in this column are calculated by assigning the reference value (the number of goals) to the highest-priority goal, which is 5 in this example, and then assigning values to the remaining goals in descending order, ending with the value of 1 for the lowest-priority goal. The weights are then summed up, and each weight is divided by the sum of all weights so that the sum of all values is 1. The following five columns represent the candidate MEPs. For each goal, the MEP's answer values can be seen as well as the final weight that the MEP receives for that goal after multiplying the answer weights by the goal weight. Each MEP's total is calculated by summing up its score for each goal.

## 5.3 Validation of the Research Objectives

By observing the framework samples presented above, it can be noted that the DASP framework achieves its goal of formalizing the different aspects of the tailoring process presented in the case study.

The different levels of pain on the pain scale were derived from the student responses to the survey questions. When new practices were introduced to the process many students reported experiencing pain but attributed it to being inexperienced with the software practice and acknowledged that additional time and experience using the practice would alleviate the pain. Some students were unwilling to tolerate the pain at certain points in the process and deviated from the process. Others tolerated the pain but reported that they would tailor the process in the next iteration by replacing one practice with another. Since different students have different experiences and different levels of tolerance for the same pain point, the DASP framework provides personalized tailoring for each software developer even if the initial process they start out with is the same.

The design of the catalog entry provides software developers with insight into the available software practices by describing the detailed steps and information required to successfully use each practice. It also provides a visual comparison of the practice being viewed to the available candidate practices in terms of their achievement of developer goals.

The initial developer survey in the DASP framework provides software developers with an opportunity to reflect on their goals for a specific software activity and prioritize them based on what they value most. It also records the developer's experience with each of the candidate practices to be able to provide more accurate results when evaluating practices based on a specific pain point. The expert knowledge that was used to make recommendations in the case study has been replaced with an automated framework that interacts with each developer individually and evaluates the same practices differently for each developer based on his/her experiences and goals. Rather than having to test multiple practices simultaneously to find the most suitable one for the context as some students opted to do in the case study, developers are presented with a practice that will most likely achieve their activity goals. The catalog entries for the practices allow developers to compare practices side by side and compare their achievement of activity goals to make informed decisions with the aid of the framework.

# Chapter 6

# Conclusions and Future Work

## 6.1 Summary

This research has presented a novel approach to adjusting a personal software process in a responsive, effective, and systematic manner based on a software developer's pain points. When it comes to tailoring software process, the conventional school of thought suggests tailoring software practices at the conclusion of a project in preparation for the next, focusing benefit on future projects. The contemporary agile philosophy is to alter a software process as soon as it is perceived to be dysfunctional, which lacks order and structure, and basically relies on intuition and hunches. The tailoring approach introduced provides a middle ground in which software practices are adjusted while a project is underway and are adjusted in a disciplined fashion.

This research describes the components of a comprehensive Dynamically Adjustable Software Process (DASP) tailoring framework, and how they interact with one another to provide a dynamic and customized personal software process built to best fit a software developer's needs and experience. As a first step toward achieving the big picture, the following contributions have been made:

1. Defining a format for cataloging representative MEPs. This catalog documents key aspects and characteristics of MEPs crucial to successfully tailoring a software process.

2. Describing a method for measuring and representing pain points. This helps software developers communicate the level of pain they are experiencing with an MEP, and helps determine when tailoring is required.

3. Introducing a framework for providing automated help in resolving pain points. The DASP framework was designed to make tailoring recommendations for the developer's current MVP based on pain points and existing MEPs. The recommendations are based on attributes and features that the developer values most in a given context or situation. As part of achieving this objective, a methodology for evaluating and selecting an alternate MEP from a number of candidate practices is described in detail.

In order to gain insight into the value of building the DASP framework, an investigation into software developers' attitudes toward tailoring a personal software process based on their pain points was carried out in the form of a case study. The case study was also a valuable way to gather any feedback that would help better achieve the research contributions mentioned above. The case study participants included 69 graduate and senior undergraduate students enrolled in two sections of the Software Process (COMP5700/COMP6700/COMP6706) course offered by the Department of Computer Science and Software Engineering at Auburn University during the Fall 2015 semester. Students that were enrolled in the course voluntarily offered feedback regarding the course assignments anonymously. The case study was guided by the following questions:

1. How do students perceive the concept of tailoring a software process based on their pain points?

2. How likely are students to adopt this method of tailoring software process outside the Software Process classroom?

A survey was administered at the end of the Fall 2015 semester and contained eleven questions regarding the Software Process assignments throughout the semester, in which students were assisted in tailoring their software process based on their pain points.

The case study rendered positive results and revealed that a majority of the students were supportive of the concept of tailoring software process based on pain points, and would likely use this concept for their future projects.

When asked whether adjusting a software process based on pain points improved their adherence to the software process, 69% of the students believed that it did improve their adherence, 21% were neutral, and only 10% of the students did not believe that it improved their adherence to the software process.

When asked whether adjusting a software process based on pain points was in line with their personal styles of software development, 62% of students said that this method of tailoring was in line with their personal style, 28% were neutral and only 10% of the students said that it was not in line with their personal style.

When asked whether they are likely to tailor software process based on their pain points outside the Software Process course, an average of 82% said they were likely to do so, 14% were neutral and only 4% of the students said they were unlikely to do so.

In addition to these extremely encouraging results, other questions on the survey validated the need for an MEP catalog that would have better assisted students in selecting between candidate Analysis MEPs when given the option to do so. The need for a method to represent pain and communicate the level of pain was also validated, as results from some of the questions revealed that in some cases, despite students reporting newly introduced aspects of the process as painful, they were willing to tolerate the pain because they saw benefit in using them. An example of this is the TDD MEP that 56% of students reported as painful in the iteration it was introduced in, but 100% of students said they were likely to use this MEP in the future outside the classroom.

In order to achieve the research contributions mentioned above, the DASP framework was described in detail, and demonstrations were made to show how it can assist software developers in tailoring their software processes based on pain points they are experiencing. The DASP framework evaluates candidate MEPs and makes recommendations for replacing the painful MEP based on a software developer's experience and goals.

## 6.2 Future Work

Much progress has been made toward achieving a comprehensive DASP framework, but much still lies ahead. The following recommendations are offered for expanding the work that has been described in this research:

1. **Building an MEP catalog** - now that an MEP catalog entry format has been designed, the next step would be to gradually build a catalog of MEPs for each of the MSAs in PCSE. This would make all the information relevant to tailoring readily available and easily accessible to software developers using the DASP tailoring framework. The MEP catalog would aid in better decision making when introducing a new MEP into a software process, especially for agile practices where many MEPs lack documentation.

2. **Automating the framework** - as can be noted from many of the student responses in the case study, tool availability and automation play a major role in software developers' willingness to use a software process. Tailoring a software process is no different, and the next step for the DASP framework would be to provide an automated solution that links the Developer Survey to the analysis tool to the MEP catalog entries, to ensure that the developer receives the maximum benefit possible from the framework with minimal overhead.

3. **Defining how long a developer must wait before reporting a newly introduced MEP as a pain point** - it was observed in the case study that each time an MEP was introduced into the software process it was reported as the most painful aspect, and as iterations passed the pain point was reported less and less. When asked what they recommended to alleviate pain most students recommended more practice. Despite the fact that most software developers are aware of the pain associated with newly introduced MEPs and would report it as tolerable pain until they gained proficiency with the MEP, we must find a way to determine how much time/how many

iterations must pass between introducing each MEP and allowing a software developer to report it as a pain point that requires tailoring. We must also define exceptions since not all pain points are alleviated by more practice. This will prevent developers from entering an endless loop of introducing an MEP, reporting it as painful, and replacing it each iteration.

Bibliography

[Abrahamsson, 2002] Abrahamsson, P. (2002). *Agile Software Development Methods: Review and Analysis (VTT publications)*.

[Abrahamsson et al., 2003] Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 244–254. Ieee.

[Alliance, 2013] Alliance, A. (2013). Practices: timeline. http://guide.agilealliance.org/timeline.html. [Online; accessed 22-September-2015].

[Avison and Fitzgerald, 1995] Avison, D. E. and Fitzgerald, G. (1995). *Information systems development: methodologies, techniques and tools*. McGraw-Hill.

[Bailey et al., 2003] Bailey, E., Card, D., Dean, J., Hall, F., Jones, C., Layman, B., and Mcgarry, J. (2003). Practical software and systems measurement guide. *DoD and US army*.

[Banerjee, 2012] Banerjee, U. (2012). Brief history of agile movement. https://setandbma.wordpress.com/2012/03/23/agile-history/. [Online; accessed 22-September-2015].

[Basili, 1992] Basili, V. R. (1992). Software modeling and measurement: the goal/question/metric paradigm.

[Beck, 2000] Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

[Beck and Andres, 2004] Beck, K. and Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. Pearson Education.

[Beck et al., 2001] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development. [Online; accessed 22-September-2015].

[Boehm and Turner, 2003] Boehm, B. and Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*. Addison-Wesley Professional.

[Brooks, 1987] Brooks, F. (1987). *No silver bullet*. April.

[Brown, 2007] Brown, S. (2007). Standardized technology evaluation process (step) users guide and methodology for evaluation teams.

[CMMI, 2001] CMMI (2001). Capability maturity model integration (cmmism), version 1.1 cmmism for systems engineering and software engineering (cmmi–se/sw, v. 1.1) continuous representation. *Continuous Representation, CMU/SEI-2002-TR-001, ESC-TR-2002-001.*

[Conboy and Fitzgerald, 2007] Conboy, K. and Fitzgerald, B. (2007). The views of experts on the current state of agile method tailoring. In *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, pages 217–234. Springer.

[Dudziak, 1999] Dudziak, T. (1999). extreme programming an overview. *Methoden und Werkzeuge der Softwareproduktion WS*, 2000.

[Fuggetta, 2000] Fuggetta, A. (2000). Software process: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 25–34. ACM.

[Gallivan, 2003] Gallivan, M. J. (2003). The influence of software developers creative style on their attitudes to and assimilation of a software process innovation. *Information & Management*, 40(5):443–465.

[Goyal, 2008] Goyal, S. (2008). Major seminar on feature driven development.

[Hafterson, 2011] Hafterson, T. (2011). Incorporating agile methods into the development of large-scale systems. In *UMM CSsci Senior Conference, Moris, MN.*

[Henninger, 2001] Henninger, S. (2001). Turning development standards into repositories of experiences. *Software Process: Improvement and Practice*, 6(3):141–155.

[Highsmith, 2000] Highsmith, J. (2000). Retiring lifecycle dinosaurs: Using adaptive software development to meet the challenges of a high-speed, high change environment.

[Highsmith, 2002] Highsmith, J. (2002). *Agile Software Development Ecosystems*. Detective Inspector Carol Ashton mystery. Addison-Wesley.

[Humphrey, 1995] Humphrey, W. S. (1995). A personal commitment to software quality. In *European Software Engineering Conference*, pages 5–7. Springer.

[ISO, 2015] ISO (2015). Iso 9000 - quality management. http://www.iso.org/iso/iso_9000. [Online; accessed 23-September-2015].

[ISO/IEC, 2008] ISO/IEC (2008). Ieee guide–adoption of iso/iec 90003:2004 software engineering–guidelines for the application of iso 9001:2000 to computer software. *IEEE Std 90003-2008*, pages 1–89.

[ISO/IEC, 1998a] ISO/IEC, J. T. C. (1998a). Information technology: Software process assessment: Part 2: A reference model for processes and process capability. Technical Report 15504-2, ISO/IEC, ISO/IEC Copyright Office, Case postale 56, CH-1211 Genve 20, Switzerland.

[ISO/IEC, 1998b] ISO/IEC, J. T. C. (1998b). Information technology: Software process assessment: Part 7: Guide for use in process improvement. Technical Report 15504-7, ISO/IEC, ISO/IEC Copyright Office, Case postale 56, CH-1211 Genve 20, Switzerland.

[Janes et al., 2008] Janes, A., Sillitti, A., and Succi, G. (2008). Non-invasive software process data collection for expert identification. In *SEKE*, pages 191–196. Citeseer.

[Kalus and Kuhrmann, 2013] Kalus, G. and Kuhrmann, M. (2013). Criteria for software process tailoring: a systematic review. In *Proceedings of the 2013 International Conference on Software and System Process*, pages 171–180. ACM.

[Konrad et al., 1995] Konrad, M., Paulk, M., and Graydon, A. (1995). An overview of spice's model for process management. In *Proc. 5th International Conference on Software Quality*.

[Kuvaja, 1994] Kuvaja, P. (1994). *Software Process Assessment and Improvement: The BOOTSTRAP Approach*. Blackwell.

[Liddle, 2013] Liddle, G. (2013). Why agile is so popular: Are you ready? http://www.summa-tech.com/blog/2013/12/16/why-agile-is-so-popular-are-you-ready-to-be-agile. [Online; accessed 22-September-2015].

[Meso and Jain, 2006] Meso, P. and Jain, R. (2006). Agile software development: adaptive systems principles and best practices. *Information Systems Management*, 23(3):19–30.

[Mnkandla, 2008] Mnkandla, E. (2008). *A Selection Framework For Agile Methodology Practices: A Family of Methodologies Approach*. PhD thesis, Faculty of Engineering and the Built Environment, University of The Witwatersrand, Johannesburg.

[Moczar, 2013] Moczar, L. (2013). Why agile isn't working: Bringing common sense to agile principles. http://www.cio.com/article/2385322/agile-development/why-agile-isn-t-working–bringing-common-sense-to-agile-principles.html. [Online; accessed 22-September-2015].

[Radice et al., 1985] Radice, R. A., Roth, N. K., O'Hara, A., and Ciarfella, W. A. (1985). A programming process architecture. *IBM Systems Journal*, 24(2):79–90.

[Sanders, 2009] Sanders, S. (2009). Writing great unit tests: Best and worst practices.

[Schwaber, 2004] Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft Press.

[Wiegers, 2005] Wiegers, K. (2005). Software process improvement handbook: A practical guide. http://www.processimpact.com/handbooks/spi_intro_and_chapter_1.pdf. [Online; accessed 22-September-2015].

[Wikipedia, 2015a] Wikipedia (2015a). Extreme programming — wikipedia, the free encyclopedia. [Online; accessed 18-September-2015].

[Wikipedia, 2015b] Wikipedia (2015b). Feature-driven development — wikipedia, the free encyclopedia. [Online; accessed 23-September-2015].

[Wikipedia, 2015c] Wikipedia (2015c). Scrum (software development) — wikipedia, the free encyclopedia. [Online; accessed 23-September-2015].